Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza
1542

hiberus
TECNOLOGÍA

Integración del sistema de cita previa SINTRA con sistemas de información sanitarios

Pedro Allué Tamargo

Director: Sergio Ilarri Artigas

Grado en Ingeniería Informática
Sistemas de Información

Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

2021

Agradecimientos

A mi familia, amigos y compañeros. Gracias por hacer el esfuerzo de intentar entender los extraños términos que utilizaba y por estar apoyándome durante el desarrollo de este trabajo.

Resumen

Este documento describe el desarrollo de una integración de un sistema de cita previa existente con los sistemas de información sanitarios. Estos sistemas sanitarios suelen contar con un sistema de gestión de citas pero no es accesible para el público en general. Con esta integración se busca permitir que sea el público general el que pueda pedir cita en los sistemas sanitarios.
El sistema de cita previa existente se denomina SINTRA. SINTRA está desarrollado por Hiberus y posee otras funcionalidades aparte de un sistema de cita previa. También cuenta con una sección para que los agentes del mismo puedan consultar, modificar y cancelar las citas que tienen asignadas. SINTRA no está orientado a ningún contexto en concreto y por ello se plantea este proyecto como una integración de este con los sistemas de información sanitarios.

Para el desarrollo de este proyecto se ha creado un componente encargado de integrar los sistemas existentes por medio de los estándares sanitarios más utilizados: HL7 y FHIR. El sistema se ha desarrollado con Java y el framework *Spring Boot* dada su facilidad y versatilidad de uso. Para la gestión de los mensajes HL7 y FHIR se utilizan las librerías de *HAPI* y *HAPI FHIR*. Estas librerías son ampliamente utilizadas en la industria.
Además, el proyecto también ha contado con un bus de integración sanitario que permite manipular los mensajes del estándar HL7 con el objetivo de integrarse con el sistema de información sanitario. El bus de integración es un componente capaz de realizar conversiones entre los mensajes para conseguir integrarse con otros sistemas. El bus empleado es *Mirth Connect*. Se trata de un componente de código abierto con una gran comunidad detrás. La variabilidad y la gran cantidad de información disponible son los puntos fuertes de este componente.

El sistema sanitario con el que se ha realizado la integración es *OpenMRS*. Este sistema es de código abierto y es uno de los referentes. Muchos sistemas sanitarios de código abierto se basan en *OpenMRS* como un modelo a seguir. Este sistema tiene una interfaz para la comunicación con el estándar HL7 pero no todos los mensajes están soportados por este. Por lo tanto, como se ha comentado anteriormente, el bus de integración es un componente fundamental para la realización de la integración.

El desarrollo de este sistema se ha hecho utilizando metodologías ágiles. Se ha basado en la división del proyecto en iteraciones de dos semanas de duración. Con esta organización se ha permitido un desarrollo más rápido. De esta forma, se ha conseguido pasar por todas las etapas del desarrollo. Cada iteración sigue las etapas de análisis, implementación y validación. La validación del componente desarrollado se realiza mediante tests automáticos, tanto unitarios como de integración. Además, se utiliza el sistema de integración continua de GitHub para validar el funcionamiento de los cambios que se suben al repositorio remoto.

Abstract

This document shows the development of an appointment's system integration with health information systems. Health information systems usually have an appointment managing system but this system isn't reachable to whole public. With this integration work the whole public will be allowed to schedule a new appointment in health information systems.
Existing appointment's system is called SINTRA. SINTRA has been developed by Hiberus and it has more functions aside an appointment system. It also has a section so that its agents can see, modify and cancel the appointments they have assigned. SINTRA isn't focused on any specific context and so this project is proposed as an integration of this and health information systems.

In this project a new component has been developed in order to integrate existing systems using health information exchange standards such as HL7 and FHIR. The system has been developed with Java and *Spring Boot* framework due its easiness and versatility to use. To manage HL7 and FHIR messages, libraries such as *HAPI* and *HAPI FHIR* are used. These libraries are widely used in the industry.
This project also has a health enterprise service bus that allows handle HL7 messages to integrate with health information system. Enterprise service bus (*ESB*) is a component that allows handle conversions between messages to achieve integrations with other systems. The ESB used is *Mirth Connnect*. This is an open source component that has a big community under it. Variability and the big ammount of documentation about it are the strong points of this component.

Health information system with which the integration has been made is *OpenMRS*. This system is open source and it is a benchmark. A lot of health information systems has relied on *OpenMRS* as a role model. This system has an interface to communicate with HL7 standard but not all messages are supported. Therefore, as commented previously, enterprise service bus is a key component to the integration development.

Development of this system has been made by using agile methodologies. Development has been based by dividing project into two weeks duration sprints. This management has allowed a faster development. This way it has been posible to go through all stages of development. Each sprint follows analysis, implementation and testing steps. Testing has been made by using automatic testing with unitary testing and integration testing. Furthermore, GitHub continous integration system is used to test system functionality with each push to remote repository.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se recoge la información acerca del contexto y los objetivos de este proyecto. En la Sección 1.1 se habla acerca de la motivación de este trabajo y una breve introducción al sistema SINTRA. En las Sección 1.2 se trata la contextualización de este proyecto. Por último, en la Sección 1.3 se muestran los objetivos de este proyecto y las tareas necesarias para conseguirlo.

1.1. Motivación

Debido a la situación sanitaria provocada por la pandemia de COVID-19 algunos de los aspectos de la sociedad cambiaron. Aspectos como el control de aforos y la distancia de seguridad se hicieron importantes en el día a día de las personas. Por ello se desarrolló SINTRA de la mano de *Hiberus*. Este sistema es una solución dedicada a la gestión de cita previa para evitar aglomeraciones. De tal forma que los usuarios puedan reservar citas para un departamento, un servicio y una fecha dada. Este sistema es adaptable a cualquier entorno.
No obstante, ¿qué pasaría si existiera ya un sistema de citas y usuarios en el entorno que se va a implantar SINTRA?. Este es el caso de los sistemas de información sanitarios (HIS) que normalmente cuentan con un sistema de reserva de citas reservado para uso interno.
En este proyecto se trata este caso particular, la integración de SINTRA con los sistemas de información sanitarios, de cara a permitir que ambos sistemas se comuniquen entre sí. Con esta integración se permitirá reservar una cita, modificarla y cancelarla en el sistema HIS desde el sistema SINTRA.

1.2. Contexto

Este proyecto se ha desarrollado en la empresa *Hiberus Advanced Solutions*. Esta empresa se dedica tanto al desarrollo de proyectos propios como SINTRA como a la consultoría o al análisis de datos para empresas. Se puede encontrar más información acerca del sistema SINTRA en el Anexo A.
Este proyecto nace de la necesidad de obtener un sistema de cita previa que sea capaz de integrarse con los sistemas de información sanitarios.

Para la integración con un sistema sanitario se va a utilizar el HIS de OpenMRS[1] que se distribuye bajo una licencia: *Mozilla Public License v. 2.0*. En el Anexo B se puede encontrar una comparación de distintos sistemas HIS y la justificación de la elección del mismo.

1.3. Objetivos y tareas

En esta sección se recogen los objetivos y tareas del proyecto realizado. En la Sección 1.3.1 se encuentran los objetivos que debe satisfacer la integración, es decir, las características que debe soportar. En la Sección 1.3.2 se dividen los objetivos de la sección anterior en tareas.

1.3.1. Objetivos

El objetivo de este proyecto es obtener una integración del proyecto SINTRA con los sistemas de información sanitarios. Esta integración debe permitir reservar una cita, modificar la reserva de una cita y cancelar una cita en el sistema sanitario mediante el sistema SINTRA. Además también se propone la creación de unos pases de acceso que puedan ser verificados en unos tornos automatizados en la entrada del centro sanitario donde tenga lugar la cita.
Además para facilitar futuras integraciones este sistema va a utilizar los estándares de mensajería sanitario HL7 y FHIR. También se pretende utilizar un bus de integración del ámbito sanitario para mejorar la interoperabilidad entre las distintas versiones del estándar HL7 y facilitar la integración con otros sistemas.

1.3.2. Tareas

Las tareas realizadas para alcanzar los objetivos son las siguientes:

- **Comprensión del estándar HL7 y FHIR:** Mediante la comprensión del funcionamiento de estos estándares de mensajería.
- **Comprensión del HIS a integrar:** Mediante la realización de pruebas con el sistema *OpenMRS*.
- **Análisis funcional del proyecto:** Análisis de los requisitos necesarios para llevar a cabo la integración.
- **Configuración del entorno de desarrollo:** Creación del entorno de desarrollo con los componentes necesarios para llevar a cabo la integración.
- **Diseño e implementación del módulo de consultas de horarios disponibles:** Desarrollo del módulo que expone un método para la consulta de horarios disponibles para la reserva de citas en el sistema sanitario.
- **Diseño e implementación del módulo de citas:** Desarrollo del módulo que expone métodos para la reserva, modificación y cancelación de citas en el sistema sanitario.

[1] `https://openmrs.org/`

- **Diseño e implementación del módulo de pases de acceso:** Desarrollo del módulo que expone métodos para la consulta y verificación de pases de acceso por parte del torno situado a la entrada del centro sanitario.
- **Integración del sistema desarrollado con SINTRA:** Desarrollo de la integración del sistema con el sistema existente SINTRA.

El desarrollo se ha dividido en distintas iteraciones siguiendo metodologías ágiles. En el Capítulo 4 se puede encontrar más información acerca de la metodología utilizada. En la Figura 4.1 se muestra la distribución temporal de las tareas a realizar mediante un diagrama de Gantt.

En los posteriores capítulos de este documento se puede encontrar la descripción del trabajo realizado. En el Capítulo 2 se puede encontrar el análisis funcional del proyecto a realizar. En el Capítulo 3 se encuentra el análisis técnico del proyecto descrito, su división en componentes y los módulos que lo componen. En el Capítulo 4 se comenta la metodología empleada, su división en iteraciones, las fases realizadas en cada una de ellas y las tecnologías utilizadas en este proyecto. En los Capítulos 5, 6 y 7 se tratan los detalles de implementación de los módulos del componente *Integrador*. En el Capítulo 8 se habla de la integración del sistema original con el componente *Integrador* desarrollado. Por último el Capítulo 9 trata sobre las conclusiones del trabajo, tanto personales como técnicas, y sobre las perspectivas de futuro e implantación del proyecto.

Capítulo 2

Análisis Funcional

En este capítulo se va a realizar un análisis funcional de los requisitos del sistema a desarrollar. Al tratarse de una integración los requisitos vienen dados de las funcionalidades de SINTRA que se van a integrar con el sistema de información sanitario. En la Sección 2.1 se tratarán los requisitos del sistema de acuerdo a los objetivos del mismo (Sección 1.3.1). En la Sección 2.2 se mostrarán los casos de uso del sistema. Por último, en la Sección 2.3 se presentará el diccionario de términos utilizados junto con su definición.

2.1. Requisitos del sistema

A continuación se van a exponer los requisitos del sistema a desarrollar en forma tabular. En la Sección 2.1.1 se encontrarán los requisitos funcionales del sistema mientras que en la Sección 2.1.2 se encontrarán los no funcionales.

2.1.1. Requisitos funcionales del sistema

Identificador	**Descripción**
RF-1	El sistema debe permitir a los usuarios reservar una cita en el sistema de información sanitario en una fecha y hora determinadas.
RF-2	El sistema debe verificar los pases de acceso de los usuarios y sus acompañantes según los criterios de que se corresponda con una cita reservada para el día y hora actuales, y disponga de usos suficientes.
RF-3	Los trabajadores podrán cancelar las citas reservadas por otros usuarios.
RF-4	Los trabajadores podrán modificar las citas reservadas por otros usuarios.
RF-5	Los trabajadores podrán pedir citas en nombre de otros usuarios.
RF-6	El sistema permitirá seleccionar el número de acompañantes cuando se reserve una cita.

2.1.2. Requisitos no funcionales del sistema

Identificador	Descripción
RNF-1	El sistema se comunicará con los sistemas de información sanitarios utilizando los estándares HL7 v2.5 (separado por tuberías) y FHIR.
RNF-2	El sistema funcionará como un componente conectable al sistema SINTRA de forma que se pueda conectar y sea independiente del sistema original.

2.2. Casos de uso

El diagrama de casos de uso del sistema se puede observar en la Figura 2.1. Se puede observar que existen dos tipos de actores en el sistema. Los usuarios son actores que quieren reservar una cita en el sistema y a los cuales se les otorga un pase de acceso para acceder al centro. El otro actor del sistema son los trabajadores que pueden realizar las mismas acciones que los usuarios y además pueden modificar y cancelar las citas ya existentes.

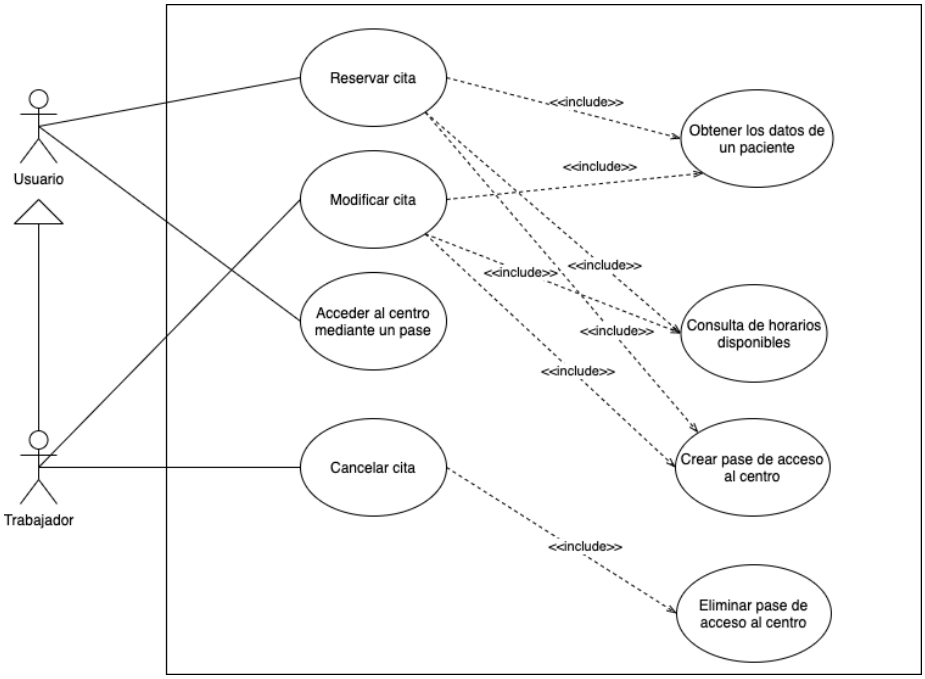

Figura 2.1: Diagrama de casos de uso del sistema a desarrollar

2.3. Diccionario de datos

A continuación se van a presentar las definiciones de los términos utilizados en los requisitos y en los casos de uso. Esta terminología será utilizada reiteradamente a lo largo del proyecto. ***Usuario*** es un actor que utiliza el sistema para la petición de una cita. Este usuario deberá tener una correspondencia con un paciente en el sistema de información sanitario en el que se quiere pedir cita. ***Acompañante*** es una persona que acompaña a un usuario que tiene una cita reservada. ***Trabajador*** se trata de un actor del sistema que se corresponde con un profesional del sistema sanitario. ***Centro*** es el lugar o centro médico donde se atienden a los pacientes que reservan citas. ***Paciente*** es una persona del sistema de información sanitario que actúa como interesado en la petición de citas. ***Cita*** se corresponde con una reunión entre un paciente y un profesional sanitario en una ubicación (centro) dada en una fecha y hora determinadas. ***Pase de acceso*** se trata de un permiso de acceso asignado a un paciente para poder acceder a un centro. Se solicita mediante el sistema SINTRA y permite el acceso al centro en una fecha y horas determinadas. Además estos pases tienen asociado un número de usos para acceder al centro. ***Permiso de acceso*** es un código QR que se obtiene si se ha reservado una cita.

Capítulo 3

Análisis Técnico

A continuación se va a proceder a analizar los requisitos técnicos del sistema a desarrollar. En la Sección 3.1 se tratan las cuestiones sobre el diseño del sistema a nivel de los componentes que lo forman. En la Sección 3.2 se describen a alto nivel las interfaces que debe tener cada componente del sistema. En la Sección 3.3 se presentarán los tipos de mensajes utilizados para llevar a cabo esta integración.

3.1. Diseño del sistema

La decisión de extraer la lógica de la integración a un *middleware* externo se basa en intentar maximizar la propiedad de la escalabilidad. Al tratarse de una integración con un conjunto de sistemas específicos (sistemas de información sanitarios) esta decisión permite entender la integración como un componente que se podrá incluir en el sistema siguiendo la línea de la descomposición en microservicios.
También la extracción de la funcionalidad a un componente externo mantiene la arquitectura por capas existente en SINTRA y da una independencia de la integración sobre el sistema ya existente.

En la Figura 3.1 se puede observar la distribución de los componentes del sistema. Se pueden distinguir los siguientes componentes:

- *Integrador:* mantiene la lógica de la integración. Ofrece APIs para el consumo por parte del sistema SINTRA y consume las APIs del sistema de información sanitario a integrarse.
- *ESB (Enterprise Service Bus):* Bus de integración orientado al ámbito sanitario. Se encargará de las conversiones entre las distintas versiones de HL7, así como el procesamiento de los mensajes que no se pueda encargar el HIS.
- *HIS:* Sistema de información sanitario con el que se quiere realizar la integración.
- *SINTRA:* Sistema de información de cita previa existente.

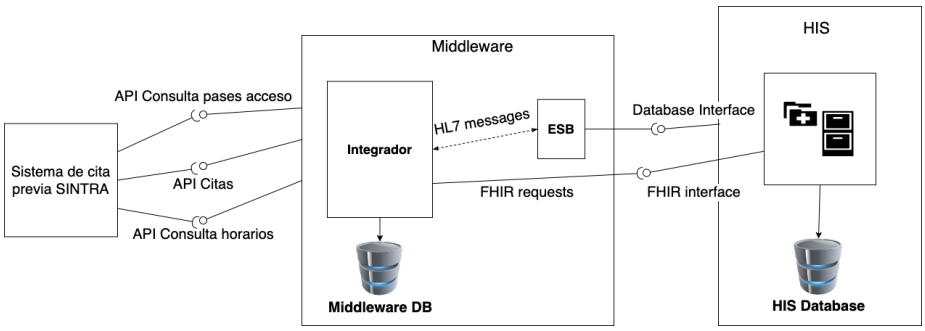

Figura 3.1: Diagrama de componentes del sistema a desarrollar

3.2. Descripción de las interfaces de los componentes

Se ha podido observar que el sistema se divide en 4 componentes: el sistema origen (SINTRA), el sistema a integrar (HIS), el componente *Integrador* y el bus de integración (ESB). En esta sección se va a hacer un recorrido por ellos y se explicará de una forma breve las interfaces que tienen que tener.

El componente *Integrador* expondrá interfaces para: consultar horarios, crear citas, modificar citas y cancelar citas. Se utilizará *HTTP* como el canal de envío de mensajes. Los mensajes estarán codificados en formato *JSON*. Este componente no será accesible desde fuera de la red del sistema SINTRA. Se puede encontrar más información acerca de los módulos de horarios y de citas en los Capítulos 5 y 6, respectivamente.

El bus de integración (ESB) expone las interfaces necesarias para llevar a cabo la integración a partir de los mensajes HL7 del componente *Integrador*. Es el encargado de comunicar los mensajes HL7 al sistema HIS y construir los mensajes de vuelta para el componente *Integrador*. Los mensajes HL7 se enviarán vía *HTTP* a este componente. Este componente no será accesible desde fuera de la red del sistema SINTRA. Para más información acerca del funcionamiento del bus de integración se puede consultar el Anexo D.

El sistema HIS es el sistema sanitario con el que se quiere realizar la integración. Este sistema debe tener una interfaz FHIR. Además el sistema deberá exponer una interfaz JDBC para acceder a su base de datos a fin de poder realizar la integración. En el Anexo B se explica el porqué es necesaria esta interfaz.

El sistema SINTRA es el encargado de gestionar la parte visual de la aplicación de cita previa y de comunicarse con el componente *Integrador*. Este sistema expone una *API Restful* para permitir que la aplicación web se comunique con el servidor que proveerá los datos necesarios para su funcionamiento. Estos datos podrán encontrarse en la propia aplicación SINTRA o en el sistema HIS. En el Capítulo 8 se trata la integración del componente *Integrador* con este sistema.

3.3. Mensajes HL7

Los mensajes utilizados para este proyecto son los mensajes *SQM_S25* (consulta de horarios) y *SRM_S01* (reserva de citas), *SRM_S02* (modificación de citas) y *SRM_S04* (cancelación de citas). Además todos estos mensajes cuentan con su mensaje de respuesta que, respectivamente, son: *SQR_S25*, *SRR_S01*, *SRR_S02*, *SRR_S04*.

Los mensajes HL7 se dividen en segmentos, y estos segmentos a su vez, se dividen en campos. Los campos pueden ser tipos de datos o mantener una división en componentes. Los caracteres de separación de los segmentos se puede encontrar en el primer segmento (*MSH*), en el campo 2[1]. En este proyecto se ha utilizado la versión 2.5 del estándar, utilizando *pipes* (|) como carácter separador de los campos del mensaje.
Además, la estandarización de los mensajes y segmentos utilizando HL7 proporciona una alta reusabilidad. Por ejemplo, en los mensajes utilizados aparece el segmento *PID* que se utiliza para identificar los datos de un usuario[2]. Esto significa que este segmento va a tener la misma estructura en todos los mensajes que aparezca, teniendo como resultado que cada campo del mismo siempre tiene el mismo significado.

En el Anexo F se puede encontrar más información sobre los mensajes utilizados en este proyecto, su estructura y sus casos de uso.

[1] `https://hl7-definition.caristix.com/v2/HL7v2.5/Segments/MSH`
[2] `https://hl7-definition.caristix.com/v2/HL7v2.5/Segments/PID`

Capítulo 4

Metodología y tecnologías de la integración

En este capítulo se va a tratar la metodología utilizada así como los temas más generales de la integración. En la Sección 4.1 se explicará la metodología aplicada en ese proyecto así como la división en distintos entregables. En la Sección 4.2 se puede encontrar una descripción de las tecnologías utilizadas tanto en el componente *Integrador* como en el bus de integración.

4.1. Metodología

Como se ha comentado en el Capítulo 1, este proyecto ha sido realizado utilizando metodologías ágiles, en este caso, SCRUM. Esta metodología implica mantener reuniones frecuentes con el cliente (en este caso la empresa *Hiberus*) y dividir el proyecto en distintos entregables. Esta división del proyecto en proyectos más pequeños permite plantear cada problema por separado y, por lo tanto, poder abordar proyectos que tienen un nivel de complejidad más alto. Cada proyecto se divide en las etapas de análisis, desarrollo y validación.
En el caso de este proyecto se ha realizado una división de las tareas en iteraciones bisemanales. Una vez a la semana tenía lugar una reunión con el cliente y al final de la iteración en cuestión se obtenía un entregable. Este entregable podía ser probado por el cliente de tal forma que validase su funcionamiento y se corrigieran (en caso de que existieran) los factores que se considerasen necesarios.
La división en entregables, y por lo tanto en iteraciones ha sido la siguiente:

- Módulo de consulta de horarios.
- Módulo de citas: reserva de citas.
- Módulo de citas: modificación y cancelación de citas.
- Módulo de pases de acceso.
- Integración con el sistema SINTRA.

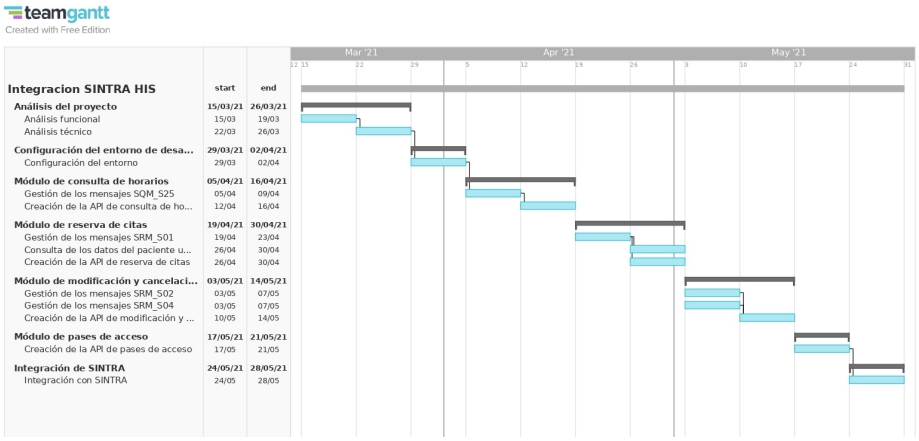

Figura 4.1: Diagrama de Gantt de la planificación del proyecto

En la Figura 4.1 se puede observar la planificación del proyecto en base a estas iteraciones. Como se ha comentado, cada iteración cuenta con las fases de: análisis, desarrollo y validación. Para la fase de validación del componente *Integrador* se han utilizado las librerías de *JUnit5*, *Spring Boot Starter Test*. Se han creado tanto *tests* unitarios como de integración. El entorno de pruebas es igual al entorno de desarrollo ya que, como se comentará posteriormente, se ha utilizado *Docker* para el desarrollo y validación del sistema. El entorno de pruebas debe tener los pacientes *Pedro Allué*, *Juan José Tambo* y *Raúl Sipán* creados con sus datos (dirección, fecha de nacimiento) bien establecidos. Además el sistema debe contar con un profesional llamado *Alfonso Verde* que además atienda citas médicas del servicio *General Medicine* el día 7/4/2021 de 11:00 a 13:00. Esto es necesario a que estos son los usuarios para los que se realizarán los *tests* de integración del sistema. Además, como punto adicional, al utilizar el sistema de control de versiones *Git*[1] con un repositorio en *GitHub*[2] se ha integrado con el sistema de *CI (Continous Integration)* de la plataforma. Con este sistema con cada *push* al repositorio se ejecutan los *tests* unitarios creados. No se han ejecutado los *tests* de integración ya que el sistema necesita que se cumplan los requerimientos explicados anteriormente. Los tests de integración se ejecutan en el entorno local sin el uso de ninguna herramienta adicional, solamente el entorno de desarrollo utilizado.
En el caso de la validación del componente *SINTRA* no se han creado tests automatizados ya que el sistema original no contaba con ellos. La validación ha sido manual, probando los distintos casos posibles mediante herramientas como *Postman*[3] en el caso de la parte del *backend*. Para el *frontend* de *SINTRA* se ha realizado una validación manual probando las distintas alternativas y, adicionalmente, el cliente ha validado su funcionamiento. En el Anexo C se muestra una descripción más detallada de las pruebas realizadas sobre el sistema.

[1] `https://git-scm.com/`
[2] `https://github.com/`
[3] `https://www.postman.com/`

4.2. Tecnologías utilizadas

Se ha utilizado *Java* como lenguaje de programación para la integración. Además se ha utilizado el *framework* de *Spring Boot* para la creación de la *API REST* de este componente. El entorno de desarrollo utilizado ha sido *IntelliJ IDEA*[4] en su versión *Community*. El uso de este entorno de desarrollo se basa en que se este entorno se ha utilizado con anterioridad y provee herramientas que simplifican algunas tareas, como sería el caso del *testing*.
Además la decisión de utilizar *Java* está respaldada por la librería utilizada para el manejo de los mensajes HL7 y FHIR. La librería utilizada para el manejo de los mensajes HL7 es HAPI[5] cuyo uso es muy extendido en la industria de la integración de sistemas sanitarios. Para el manejo de los mensajes FHIR se utiliza la librería *HAPI FHIR*[6]. Para el envío de los mensajes vía *HTTP* se ha utilizado la librería *Apache HttpClient*[7] ya que proporciona una interfaz muy simple para el envío de mensajes en este canal.
Además como parte del desarrollo se ha documentado con *Swagger* la *API REST* utilizando la librería *springdoc-openapi*[8].

El bus de integración elegido es *Mirth Connect*. Este sistema se distribuye bajo una licencia pública de Mozilla (MPL). Se ha elegido este componente ya que es *Open Source* y además muy extendido en el sector sanitario. Otro factor a tener en cuenta es que detrás de este sistema hay una amplia comunidad que ha responde a las preguntas que plantean los usuarios de la misma.
Otras opciones de bus de integración podrían ser *Rhapsody*, *Cloverleaf*, *Mulesoft*. Por ejemplo, el bus de integración *Rhapsody* se utiliza en el sistema sanitario de Aragón. No obstante, estos sistemas tienen licencias privativas y su distribución no es gratuita. Por lo tanto se ha decidido utilizar *Mirth Connect* ya que es gratuito. En el Anexo D se detalla el funcionamiento del bus de integración elegido.
Para llevar a cabo la integración con el sistema HIS elegido se ha hecho contra la base de datos. Esto es así ya que el sistema HIS no cuenta con los manejadores necesarios para gestionar los mensajes HL7 elegidos para la integración [1].

Para el desarrollo y despliegue del sistema se ha utilizado *Docker*. La decisión de uso de este sistema basado en contenedores es que permite la creación del entorno de desarrollo de una forma más ágil. Además utilizando este sistema se erradican los problemas del despliegue ya que el entorno de desarrollo y el de producción no difieren.

[4] https://www.jetbrains.com/es-es/idea/
[5] https://hapifhir.github.io/hapi-hl7v2/
[6] https://hapifhir.io/
[7] https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient/4.5.13
[8] https://springdoc.org/

Capítulo 5

Módulo de consulta de horarios

En este capítulo se va a tratar el módulo de consulta de horarios disponibles. Esta consulta se utilizará a la hora de reservar una cita para conocer los horarios disponibles dado una fecha, un profesional y el servicio que se quiere consultar. En la Sección 5.1 se presenta la interfaz de este módulo. En la Sección 5.2 se explican los aspectos de la implementación de este módulo. Por último, en la Sección 5.3 se explican los problemas encontrados durante el desarrollo de este módulo junto con la solución utilizada.

5.1. Interfaz expuesta

La interfaz expuesta por este componente se trata de una *API Restful*. Dispone de un método tipo *GET* que recibe parámetros en la URL. Los parámetros que reciben son:

```
servicioid: number;
servicionombre: string;
serviciolcoation: string;
profesional: string; (nombre del profesional)
fecha: string; (en formato yyyy-MM-dd)
```

Si no hay errores se devuelve un código *HTTP 200* y un *JSON* con la forma:

```
{
    servicio: {
        id: number;
        nombre: string;
        location: string;
    };
    profesional: string;
    fecha: string (con formato YYYY-MM-DD);
    horariosDisponibles: string[]; (hora con formato HH:
       mm)
```

```
}
```

Si alguno de los parámetros de entrada no es válido el sistema devolverá un código *HTTP 400* con un *JSON* con la forma:

```
{
    error: string;
}
```

5.2. Desarrollo del módulo

En esta sección se muestran las claves del proceso de implementación del módulo de consulta de horarios. La implementación de este módulo responde al diagrama de la Figura E.1. En la Sección 5.2.1 se trata el proceso de desarrollo del módulo en el componente *Integrador* mientras que en la Sección 5.2.2 se trata la implementación del mismo en el bus de integración.

5.2.1. Componente Integrador

Para el desarrollo del módulo se ha creado un controlador que se corresponde con la ruta `/horarios` y cuyo único método se corresponde con la petición *GET* descrita en el apartado anterior.
Como se puede observar en la Figura 5.1 se ha optado por una descomposición del problema en clases que tienen una funcionalidad acotada. De esta forma, mediante la inyección de dependencias que permite el *framework* de *Spring Boot*, se pueden aplicar los conceptos de arquitectura hexagonal [2] y aislar la lógica de negocio de la infraestructura del proyecto.
Como se ha comentado antes, en la Figura 5.1 se encuentra el diagrama de clases que utiliza el módulo de consulta de horarios disponibles. El encargado de exponer la API de consulta es el controlador, con el método de `consultaHorarios`. Por otra parte se puede observar que para el manejo y confección de los mensajes HL7 se ha utilizado el patrón *Abstract Factory* [3]. Con este patrón se permite un menor acoplamiento entre el código de creación de los mensajes y el de gestión de la petición de la API. La clase `HorariosService` se encarga de enviar el mensaje HL7 al bus de integración y recibir su respuesta.

Tras la creación del código del módulo se han creado *tests* automatizados tal y como se comenta en el Capítulo 4. Se han realizado tanto tests unitarios para comprobar el funcionamiento de las clases de la aplicación como tests de integración para el controlador. En el caso de los tests de integración se ha partido de un entorno de pruebas en el cual existía un profesional con unos huecos disponibles para un servicio y localización dados.

5.2.2. Bus de integración

Los mensajes HL7 utilizados para la consulta de horarios son: *SQM_S25* (consulta) y *SQR_S25* (respuesta). Para escuchar los mensajes enviados al canal de *Mirth* por *HTTP* se ha utilizado el componente *HTTPListener* en el

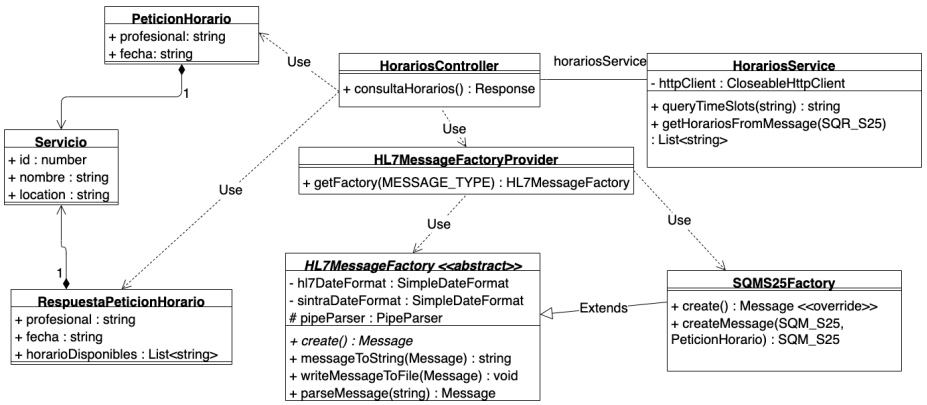

Figura 5.1: Diagrama de clases usadas en el módulo horarios

puerto 80 y en el *path* `/horarios/`. Se ha establecido la respuesta en la respuesta procesada por el componente destino, encargado de realizar acciones sobre el sistema HIS.
Para obtener la información acerca de los horarios disponibles se ha creado un transformador que mediante con la siguiente consulta SQL se encarga de devolver los horarios disponibles. Las tablas de la base de datos se pueden encontrar en la Figura 5.2.

```
select start_date, end_date
from appointmentscheduling_time_slot
where appointment_block_id in (
  select appointment_block_id
  from appointmentscheduling_appointment_block
  where provider_id = (
    select provider_id
    from provider
    where person_id = (
      select person_id
      from person_name
      where CONCAT(given_name, ' ', family_name) LIKE '
          nombreprofesional'
    )
  ) AND location_id = (
    select location_id
    from location
    where name LIKE 'localizacionservicio'
  ) AND appointment_block_id in (
    select appointment_block_id
    from appointmentscheduling_block_type_map
    where appointment_type_id = (
      select appointment_type_id
      from appointmentscheduling_appointment_type
      where name LIKE 'nombreservicio'
```

```
      )
   )
)
AND date(start_date) = date('2021-04-01 11:00')
AND time_slot_id not in (
   select time_slot_id
   from appointmentscheduling_appointment
)
```

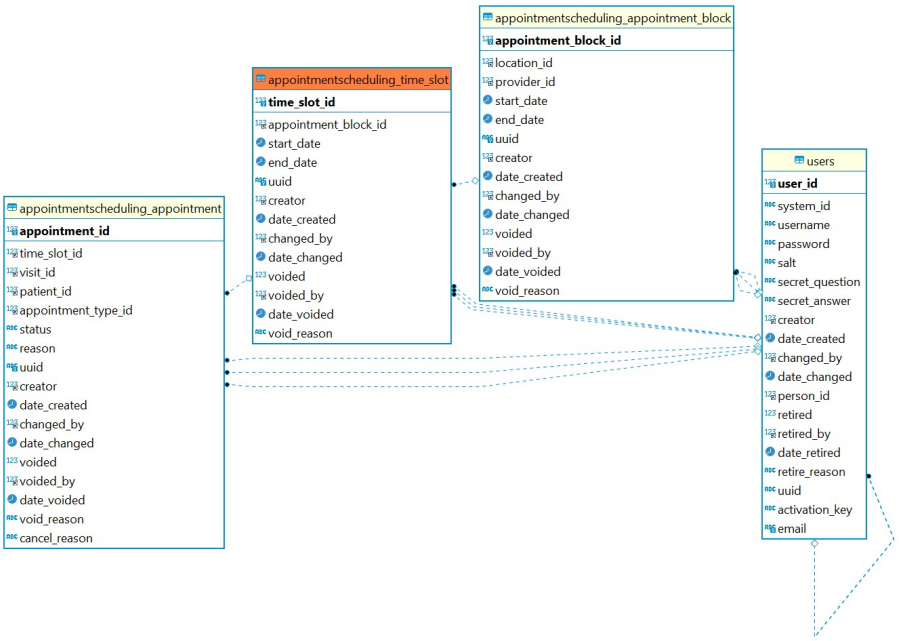

Figura 5.2: Diagrama E-R de la base de datos del HIS (horarios)

5.3. Problemas encontrados

Dado que este fue el primer componente en integrarse con el sistema HIS se presentaron varios problemas en el proceso de integración con *Mirth*. El primer problema fue que con el canal de recepción *HTTP* se tenía que construir el mensaje de respuesta utilizando cadenas de texto plano en lugar de la librería con la que cuenta el bus de integración para la construcción de mensajes. La solución a este problema pasa por utilizar los procesos de transformación de las respuestas para construir los mensajes en texto plano [4]. El segundo problema relacionado con el bus de integración viene de la complejidad de uso de una herramienta tan compleja sin tener una experiencia previa. La solución fue un estudio del manual de usuario del sistema [5] en combinación con la realización de pruebas con *Postman*.

Capítulo 6

Módulo de citas

A continuación se va a tratar el desarrollo del módulo de citas. Este es el módulo central del sistema ya que sobre este módulo se asentarán las funcionalidades de creación, modificación, cancelación y de pases de acceso. En la Sección 6.1 se presenta la interfaz que debe tener el módulo. En la Sección 6.2 se trata el desarrollo del módulo. En la Sección 6.3 se muestran los distintos problemas y las soluciones que se han tomado.

6.1. Interfaz expuesta

En esta sección se va a mostrar la *API* relacionada con el módulo de citas. En la Sección 6.1.1 se va a mostrar la API de reserva de citas. En la Sección 6.1.2 se tratará la API de modificación de citas ya existentes. Por último en la Sección 6.1.3 se mostrará la API de cancelación de citas existentes en el sistema HIS.

6.1.1. Reserva de citas

La interfaz expuesta para la reserva de citas se trata de una *API Restful* de un solo método *POST* en el path `/citas`. En el cuerpo de la petición se enviará la información en un *JSON* con el siguiente formato:

```
{
    servicio: {
        id: number;
        nombre: string;
        location: string;
    };
    profesional: string;
    fecha: string (con formato yyyy-MM-dd);
    hora: string (con formato HH:mm);
    datosPersonalesUsuario: {
        nombre: string;
        fechaNacimiento: string (con formato yyyy-MM-dd);
        address: {
```

```
            street: string;
            city: string;
            state: string;
            country: string;
        };
    };
    numeroAcompanantes: number;
    observaciones: string;
}
```

La respuesta será un código *HTTP 201* si la cita se ha creado sin problemas y se devolverá un *JSON* con la siguiente forma:

```
{
        ref: string; (uri del recurso creado)
    paseAcceso: {
        id: number;
        appointmentUUID: string;
        servicio: {
            id: number;
            nombre: string;
            location: string;
        };
        fecha: string; (con formato YYYY-MM-DD)
        horarioCita: string; (con formato HH:mm)
        finHorarioCita: string; (con formato HH:mm)
        nombreUsuario: string;
        profesional: string;
        usosRestantes: number; (siempre >=0)
    }
        message: string; ("appointment scheduled for
            $username at $date, $hour with $professional
            at $service, $location")
}
```

En caso de que ocurra algún error la respuestá tendrá un código *HTTP 400* y el *JSON* devuelto tendrá la forma:

```
{
    error: string;
}
```

6.1.2. Modificación de citas

Para la modificación de citas se expone un *endpoint* tipo *PUT* que se encuentra en el *path* `/citas/uid`, donde el parámetro *uid* se corresponde con el *UUID* de la cita creada en el sistema HIS. Se enviará un *JSON* con el siguiente formato.

```
{
    servicio: {
```

```
        id: number;
        nombre: string;
        location: string;
    };
    profesional: string;
    fecha: string (con formato yyyy-MM-dd);
    hora: string (con formato HH:mm);
    datosPersonalesUsuario: {
        nombre: string;
        fechaNacimiento: string (con formato yyyy-MM-dd);
        address: {
            street: string;
            city: string;
            state: string;
            country: string;
        };
    };
    numeroAcompanantes: number;
    observaciones: string;
}
```

En el caso de que exista un error tanto en el formato del *JSON* como en la petición al sistema HIS se devolverá un código *HTTP 400* y un *JSON* con el siguiente formato:

```
{
    error: string;
}
```

En caso de que la petición de modificación se ejecute correctamente se devolverá un código *HTTP 200* y un *JSON* con el siguiente formato:

```
{
    ref: string; (uri del recurso modificado)
    paseAcceso: {
        id: number;
        appointmentUUID: string;
        servicio: {
            id: number;
            nombre: string;
            location: string;
        };
        fecha: string; (con formato YYYY-MM-DD)
        horarioCita: string; (con formato HH:mm)
        finHorarioCita: string; (con formato HH:mm)
        profesional: string;
        nombreUsuario: string;
        usosRestantes: number; (siempre >=0)
    }
    message: string; ("appointment res-scheduled for
       $username at $date, $hour with $professional at
```

```
        $service , $location")
}
```

6.1.3. Cancelación de citas

En el caso de la cancelación de una cita se ha implementado un *endpoint* tipo *DELETE* que se corresponde al *path* `/citas/uid`. En caso de que esta petición se ejecute correctamente se devuelve el código *HTTP 200*. Si se produce algún fallo en la petición se devuelve un código *HTTP 400* con un *JSON* con el formato:

```
{
    error: string;
}
```

6.2. Desarrollo del módulo

En esta sección se va a tratar la implementación del módulo de citas. Como se ha comentado anteriormente, este módulo debe permitir la reserva, modificación y cancelación de citas. Los diagramas relacionados con la implementación de este módulo responden a las Figuras E.2, E.3 y E.4. En la Sección 6.2.1 se tratan los detalles de la implementación del componente *Integrador*. En la Sección 6.2.2 se explica el proceso de desarrollo de los canales del bus de integración necesarios.

6.2.1. Componente Integrador

El componente a desarrollar se traduce en un controlador que contendrá un método *POST* para crear las citas, un método *PUT* para modificarlas y un método *DELETE* para cancelarlas. Se ha utilizado una descomposición en clases de una forma análoga al módulo anterior (Capítulo 5) para mantener el concepto de arquitectura hexagonal.
Además en los casos de reserva o modificación de citas se tienen que obtener los datos del paciente y para ello se utiliza la interfaz FHIR del sistema HIS. De esta manera se obtiene la información del usuario en un formato estructurado [6]. Con esta información, y en especial con el identificador universal único (UUID, *Universally Unique Identifier*) se construirá el mensaje de petición de reserva de consulta. La consulta se realiza con los métodos implementados por el HIS. Esto implica que no todas las búsquedas permitidas por el estándar están soportadas por este sistema [7]. La búsqueda que se realiza tiene en cuenta el nombre completo del paciente, su apellido, su fecha de nacimiento y la dirección de su domicilio.
Cuando se reserva o se modifica una cita la cita se crea el pase de acceso con la información de la cita. El pase de acceso tendrá el mismo *UUID* que la cita almacenada en el HIS y la almacenada en el sistema SINTRA. En el Capítulo 7 se explica el módulo de pases de acceso y como se almacenan en la base de datos.
En el caso de la cancelación de una cita se comprueba que se haya expedido un

pase de acceso para esta cita y se elimina de la base de datos.
En el Anexo E se pueden encontrar los diagramas de flujo correspondientes a la implementación de estos métodos.

En la Figura 6.1 se encuentra el diagrama de clases correspondiente a este módulo. Se puede observar que el controlador `CitasController` alberga los métodos de creación, modificación y eliminación de citas. La clase `PacientesService` se encarga de realizar las consultas utilizando el estándar FHIR tal y como se ha comentado anteriormente. La clase `PaseAccesoService` tiene como objetivo el manejo de los pases de acceso que se deben generar para las citas del sistema. Además, de forma análoga a la explicada en el Capítulo 5, se ha utilizado el patrón de diseño *Abstract Factory* para la confección de los mensajes HL7 necesarios.

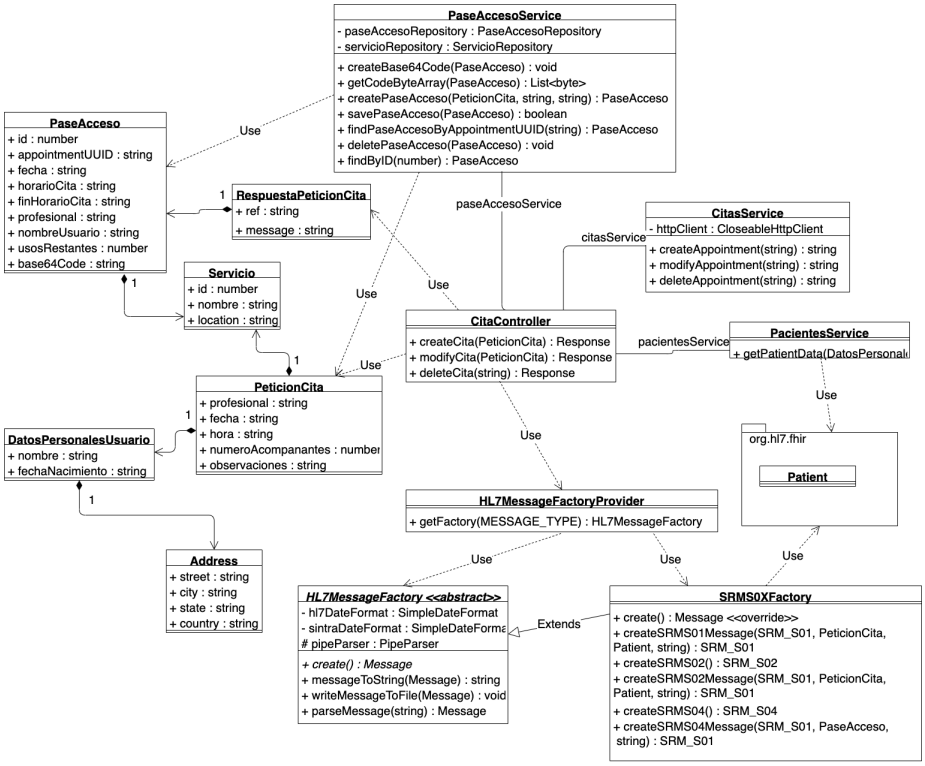

Figura 6.1: Diagrama de clases del módulo de citas

Tras la implementación del módulo se han creado *tests* de integración y unitarios para la validación del trabajo realizado. En estos *tests* se prueban los casos de la API que podrían producir fallos y se busca que cumplan con la especificación mostrada en la Sección 6.1.

6.2.2. Bus de integración

Los mensajes HL7 que se utilizan en la integración son *SRM_S0X* (petición) y *SRR_S0X* (respuesta de la petición). Como en el capítulo anterior se utiliza un componente *HTTPListener*. En el caso de la reserva de citas el componente escucha el puerto 81 y la ruta `/createCita/`. Sobre la modificación de citas el componente escucha el puerto 82 y la ruta `/modifyCita/`. Y en el caso de la cancelación de citas el componente escucha el puerto 83 y en el *path* `/deleteCita/`. La descripción detallada y el contenido de los mensajes *HL7* utilizados se puede encontrar en el Anexo F.
En el canal de reserva de citas se realiza la integración mediante una consulta del identificador del hueco solicitado (de forma análoga al capítulo anterior) y el identificador del usuario en la base de datos por medio del *UUID* del paciente. Tras obtener la información necesaria por medio de las consultas a la base de datos se realiza la inserción. Como se puede observar en la Figura 6.2 se debe realizar una inserción en la tabla de citas[1]. Además según lo observado en el comportamiento del sistema HIS las citas pasan por varios estados. El primero de ellos es *SCHEDULED*. Este sistema guarda un histórico de los estados de las citas y por ello se debe realizar una segunda inserción en la tabla de historial de las citas[2].
Además como ocurría en el Capítulo 5 la respuesta se ha de componer manualmente sin utilizar las herramientas que provee el bus de integración. Por lo tanto el estado del mensaje de respuesta variará en función del estado de las consultas a la base de datos del HIS. Como punto adicional los mensajes de error muestran información de qué ha ido mal en el procesamiento del mensaje.

En el canal de modificación de citas se utiliza una estrategia similar. Lo primero se realiza una consulta a la base de datos para obtener el identificador del nuevo hueco del horario a reservar. Seguidamente se obtiene el identificador del usuario a partir del *UUID* del mismo. Cuando se tiene toda la información necesaria para realizar el cambio del horario se realiza un borrado de las tablas comentadas y se realiza una nueva inserción en las mismas con los nuevos datos para la cita.

Por último para integrar la cancelación de citas se busca el identificador numérico de la cita a partir del *UUID* de la misma, que se encuentra en el pase de acceso creado y se eliminan sus referencias de las tablas anteriores.

[1] `appointmentscheduling_appointment`
[2] `appointmentscheduling_appointment_status_history`

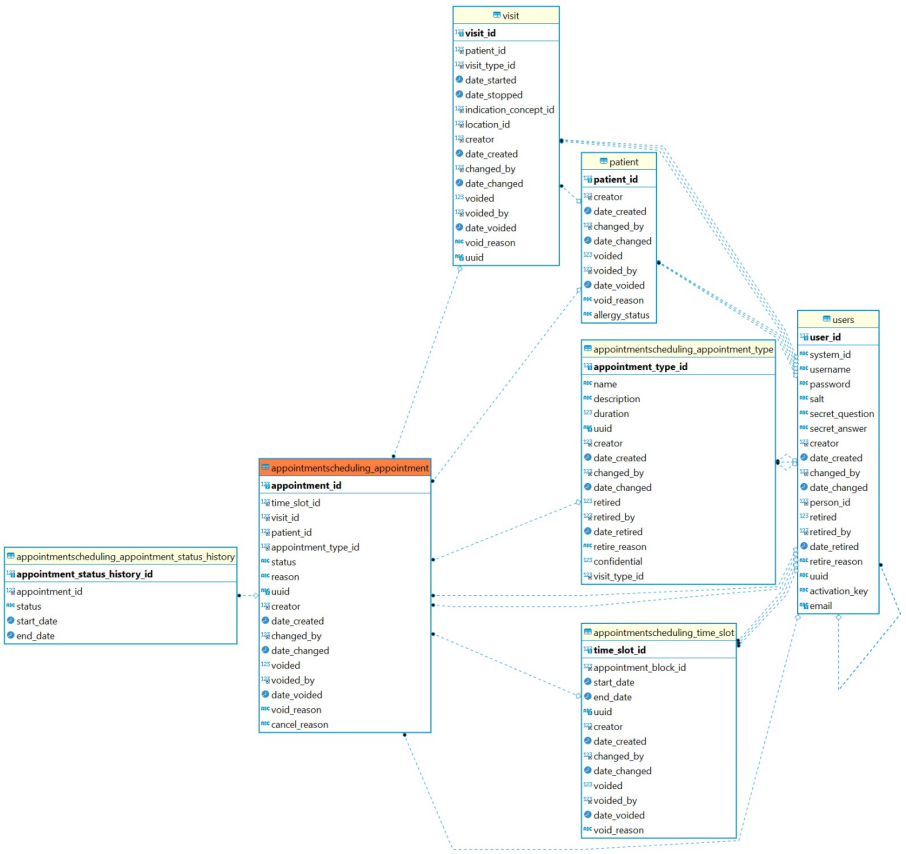

Figura 6.2: Diagrama E-R de la base de datos del HIS (citas)

6.3. Problemas encontrados

Un problema encontrado durante la implementación del componente *Integrador* es la estructura de los mensajes *SRM_S01*, *SRM_S02* y *SRM_S04*. Según el estándar estos mensajes tienen la misma forma. La implementación con la librería *HAPI* sigue el estándar. Por motivos de evitar la repetición de código esta librería no cuenta con clases para representar los mensajes *SRM_S02* y *SRM_S04* y se debe utilizar la clase *SRM_S01* para ellos. La solución es establecer el campo 9 del segmento *MSH* a los valores necesarios. En el caso de ser un mensaje *SRM_S02* sería: `SRM^ S02^ SRM_S01` y en el caso de ser *SRM_S04* sería: `SRM^ S04^ SRM_S01`.

Otro problema encontrado es que la interfaz FHIR del HIS está protegida por un mecanismo de seguridad Basic [8]. Este mecanismo provee un valor que hay que añadir a la cabecera de la petición. No obstante, al utilizar la librería de *HAPI FHIR* existe un método para autenticar la petición.

Como se ha observado en el componente del bus de integración se usa un puerto distinto al utilizado en el Capítulo 5. Esto es así ya que *Mirth* no per-

mite tener a dos canales escuchando el mismo puerto aunque se utilicen *paths* distintos. Una solución probada fue utilizar el mismo canal con el mismo puerto y redirigir los paquetes en función del *path* de la petición. No obstante, esto no funcionó ya que la respuesta de un canal debe ser seleccionada por el componente receptor (*HTTPListener*) y al utilizar mensajes de respuesta distintos en cada módulo no era una opción. Por lo tanto, se ha optado por utilizar puertos distintos para todos los tipos de mensajes. Esta solución no sería válida si el número de tipos de mensajes distintos a procesar fuera mayor.

Capítulo 7

Módulo de pases de acceso

En esta sección se va a describir el módulo de pases de acceso. Estos pases almacenan información sobre la cita reservada y se utilizan para permitir el acceso de los pacientes a los centros sanitarios. En la Sección 7.1 se presenta la interfaz que debe exponer este módulo. En la Sección 7.2 se tratan los detalles de implementación de la interfaz descrita.

7.1. Interfaz expuesta

La interfaz que expone este módulo se divide en dos métodos *GET* y un método *PUT*. El primer método *GET* se trata del correspondiente al *path* `/pases/id` y se utiliza para recuperar la información sobre un pase de acceso. Este método recibe el parámetro id del *path* de la petición. En caso de que no exista un pase de acceso con el *id* proporcionado, se devolverá un código *HTTP 404*. Si el pase de acceso existe, se devuelve un código *HTTP 200* con un *JSON* con la siguiente forma:

```
{
    id: number;
    appointmentUUID: string;
    servicio: {
        id: number;
        nombre: string;
        location: string;
    };
    fecha: string; (con formato YYYY-MM-DD)
    horarioCita: string; (con formato HH:mm)
    finHorarioCita: string; (con formato HH:mm)
    profesional: string;
    nombreUsuario: string;
    usosRestantes: number; (siempre >=0)
    base64Code: string; (codigo QR en formato base64)
}
```

El otro método *GET* se encuentra en el *path* `/pases/id/qrcode` es el que devuelve la imagen correspondiente al código QR al pase de acceso. En caso de que el pase de acceso no exista se devuelve el código *HTTP 404*. Si el pase de acceso existe se devuelve el código QR como un vector de bytes.

El método *PUT* se encuentra en el *path* `/pases/id/entrar` y su función es la de validar el pase de acceso para comprobar que existe y se puede utilizar. Si el pase de acceso con el identificador proporcionado no existe se devuelve el código de error *HTTP 404*. En caso de que el pase de acceso exista pero no se puede utilizar porque no queden usos disponibles para el mismo se devuelve un código *HTTP 400*. Si el pase de acceso existe y quedan usos restantes se devuelve un código *HTTP 200* y un *JSON* con la forma:

```
{
    id: number;
    appointmentUUID: string;
    servicio: {
        id: number;
        nombre: string;
        location: string;
    };
    fecha: string; (con formato YYYY-MM-DD)
    horarioCita: string; (con formato HH:mm)
    finHorarioCita: string; (con formato HH:mm)
    profesional: string;
    nombreUsuario: string;
    usosRestantes: number; (siempre >=0)
    base64Code: string; (codigo QR en formato base64)
}
```

Además, se modifica el número de usos restantes reduciendo en uno su valor pero este valor siempre será mayor o igual que 0.

7.2. Desarrollo del módulo

Para el desarrollo del módulo se han seguido los diagramas de las Figuras E.6 y E.7. Por ello, se ha creado un controlador que responde al *path* `/pases/`. Además, la información de los pases de acceso se debe almacenar en la base de datos para poder recuperarla posteriormente. Es por este motivo que se ha utilizado la librería de `spring-data`[1], que permite un acceso sencillo a la base de datos desde el código. Para almacenar los objetos Java se utiliza la implementación de *JPA* de esta librería y se deben anotar las clases `PaseAcceso` y `Servicio` para crear el esquema mostrado en la Figura 7.1.

Para almacenar el código QR del pase de acceso se ha utilizado la codificación en *base64* como un atributo de los objetos de la clase `PaseAcceso`. Almacenarlo como una cadena de texto en *base64* se justifica en términos de escalabilidad. Si

[1] `https://spring.io/projects/spring-data`

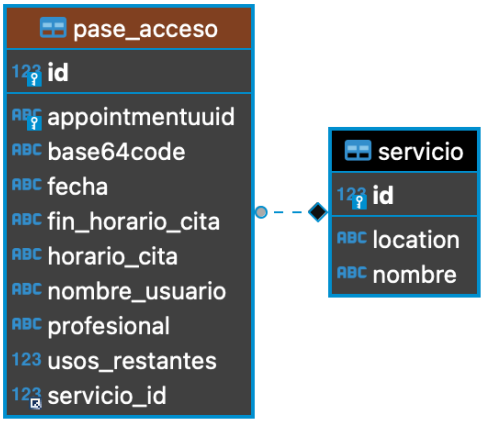

Figura 7.1: Diagrama ER de los pases de acceso

no se almacenase el código en la base de datos, habría que plantear un sistema de ficheros distribuido como *Apache Hadoop*[2] para compartir la información con los distintos nodos del componente *Integrador*.

En la Figura 7.2 se muestra el diagrama de clases del módulo de pases de acceso. Se puede observar que la clase del controlador (`PaseAccesoController`) tiene los 3 métodos descritos en la interfaz (Sección 7.1). Para acceder a la fuente de datos se utiliza la clase `PaseAccesoService` que tiene como atributos los repositorios de *Spring Data* para las clases *PaseAcceso* y *Servicio*.

[2] `https://hadoop.apache.org/`

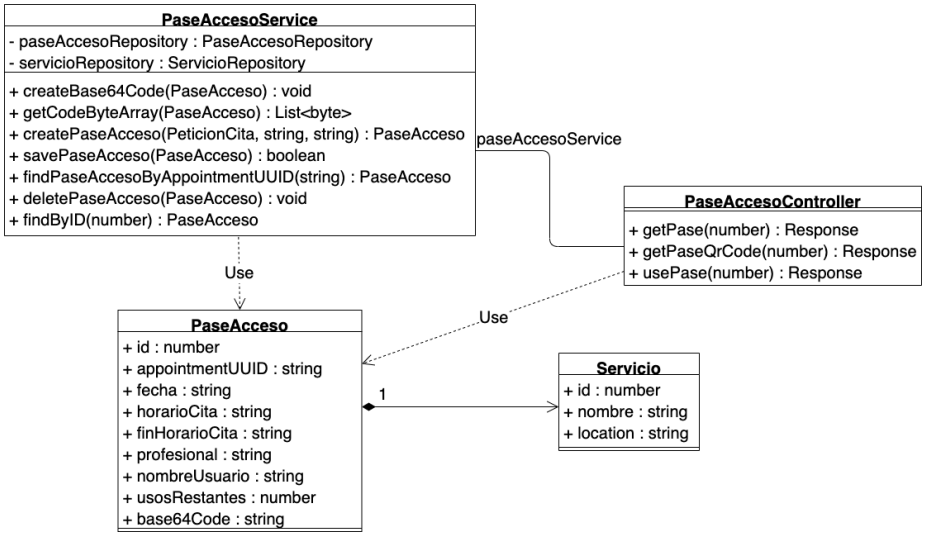

Figura 7.2: Diagrama de clases del módulo de pases de acceso

Capítulo 8

Integración con SINTRA

Como se comenta en el Anexo A las tecnologías utilizadas en este proyecto son el *framework Angular* para la aplicación web y *.NET Framework* para el servidor. En este capítulo se va a tratar la integración del componente *Integrador* con el componente *SINTRA*. En la Sección 8.1 se tratarán los aspectos y modificaciones realizadas sobre la aplicación web para integrarlo con el componente *Integrador*. En la Sección 8.2 se comentarán los aspectos realizados en el lado del servidor para realizar la integración con el componente *Integrador*.

8.1. Integración con la aplicación web

Como se muestra en la Figura A.2 la aplicación no cuenta con los datos suficientes para identificar a un paciente utilizando la interfaz FHIR del HIS (Sección 6.2.1). Por esta razón, es necesario obtener más datos sobre el usuario para realizar la petición de obtención de sus datos de usuario. Para ello, como se puede observar en la Figura A.3, se han añadido los campos necesarios para permitir al usuario introducir su fecha de nacimiento y domicilio. Adicionalmente, ya que se permite reutilizar el pase de acceso en función del número de acompañantes, también se ha añadido un campo para establecer este número de acompañantes.
De cara a un futuro trabajo de integración, tal y como se explica en el Capítulo 9, se ha añadido un campo del formulario que refleja el número de la seguridad social.
Dado que es imperativo enviar esta información al servidor, se ha modificado la clase `Appointment` que describe la estructura que deben tener los objetos de la misma.
Además, se ha sustituido el vocabulario utilizado por su equivalente del mundo sanitario. De esta forma, un servicio se traduce en un tipo de consulta (Dermatología, Neurología...) y los departamentos se convierten en las ubicaciones de los mismos. Para ello, se ha modificado el fichero de traducciones (*es.json*) que utilizaba la aplicación web[1].

[1] `https://www.npmjs.com/package/@ngx-translate/core`

8.2. Integración con el servidor

La integración del componente *Integrador* con el servidor de SINTRA se ha basado en sustituir las consultas a la base de datos local por llamadas a la API del *Integrador*. Para realizar estas llamadas se ha utilizado la inyección de dependencias de *.NET* [9]. Para ello, en la función `ConfigureServices` del fichero `Startup.cs` se ha añadido la declaración del cliente *HTTP* [10].
De igual manera que ocurría en la sección anterior, se han modificado los modelos pertenecientes al sistema original para almacenar la información enviada desde la aplicación web. Además, para permitir la comunicación con la API del componente *Integrador*, se han creado los *Data Transfer Object (DTO)* necesarios para cumplir con la especificación de la misma. Para separar la configuración del código se ha utilizado el fichero *appsettings.json* que se traduce en una clase *AppConfiguration* [11], la cual contiene la URI de la API del componente *Integrador*.

Excepcionalmente se ha tenido que modificar la API expuesta por SINTRA para acoplar el contexto del sistema al ámbito sanitario. El sistema original estaba planteado de tal forma que un usuario se encuentra asociado a un servicio del sistema. No obstante, en el sistema HIS los profesionales no están asociados a ningún servicio en concreto y, por lo tanto, se debe romper esa restricción de SINTRA. Por ese motivo, el *endpoint* utilizado por la aplicación web para la obtención de los horarios disponibles dado un servicio ha sido modificado. El nuevo *endpoint* tiene la semántica de obtener los horarios disponibles dado un servicio y un profesional del sistema.
Es por este mismo motivo por el cual se ha modificado el modelo de la base de datos del sistema SINTRA para reflejar que los usuarios del sistema no están enlazados a ningún servicio en concreto. Este cambio podría haberse evitado si se hubiera pensado que un profesional es fijo para un servicio, lo cual se hubiera traducido en restringir en el sistema HIS que solo pudiera crear huecos de consulta para un servicio dado.

Una de las funcionalidades del sistema SINTRA permite gestionar las agendas de los trabajadores. De esta forma, los trabajadores pueden obtener información sobre su agenda para un día en concreto, pudiendo usar filtros por los servicios disponibles. Para cumplir esta funcionalidad se han realizado dos pasos. El primero es almacenar las citas en el sistema SINTRA, tal y como indica el diagrama de la Figura E.2. Por lo tanto, se almacena en la base de datos del sistema la información necesaria para crear los calendarios.
El segundo paso se basa en eliminar la restricción de que los profesionales están limitados a un solo servicio. Como se ha explicado anteriormente, esto no es correcto e impedía la carga de los eventos del calendario.

Capítulo 9

Conclusiones

El desarrollo de una integración del ámbito sanitario ha sido todo un reto. Durante mi formación académica se ha estudiado la estructura de los sistemas de información sanitarios en Aragón. Con este proyecto he tenido la oportunidad de profundizar en el funcionamiento de estos sistemas y los estándares que utilizan. El uso de los estándares HL7 y FHIR permite establecer un marco común para el intercambio de información entre estos sistemas. No obstante, aunque la estructura de estos mensajes sea estándar, el contenido no lo es. Por ello, es imperativa la redacción de un documento donde se especifique la información que incluye el mensaje.

Además, el uso del bus de integración ha sido una buena solución, ya que es realmente el encargado de llevar a cabo la integración con el sistema HIS. Al ser un componente programable, se puede modificar su comportamiento para integrarlo con otros sistemas. La versatilidad de este componente es su fortaleza.

También es necesario comentar que la metodología seguida (Capítulo 4) ha sido un acierto. No había tenido la oportunidad de trabajar con iteraciones tan cortas durante la carrera y creo que es muy interesante de cara a gestionar los tiempos de un proyecto más grande. No obstante, en términos de planificación es más complejo, ya que se necesita realizar un análisis más exhaustivo de las tareas que se deben realizar para dividirlas en distintos *sprints*.

Sobre el trabajo con el sistema SINTRA es necesario comentar que su falta de documentación hacen difícil el comienzo del proyecto a un nuevo miembro. En mi caso, tuve que estudiar la estructura del código y la documentación de Microsoft [12] para hacerme una idea de cómo funcionaba el sistema.

De cara al futuro, se plantea integrar más funcionalidad del sistema HIS en SINTRA. Por el momento, los servicios y los usuarios del sistema son estáticos, es decir, inicialmente se encuentran almacenados en la base de datos de SINTRA. No obstante, se podría realizar un trabajo de carga desde el sistema HIS. Además, también se podría integrar el número de la seguridad social de cara a reducir el número de campos que debe introducir el usuario para identificarse. Sin embargo, esta mejora debe ser amparada por la interfaz FHIR del sistema HIS y por el momento no todos los campos son compatibles.

De cara a una posible implantación de este sistema, ha tenido lugar una reunión con el hospital *Sant Joan de Déu* de Barcelona. Tras esta reunión tendrá lugar una prueba de concepto para que el cliente conozca lo que puede llegar a hacer este sistema.

Como se ha comentado anteriormente, este proyecto ha sido llevado a cabo como unas prácticas en la empresa *Hiberus*. La duración de las prácticas, y por lo tanto del proyecto, se encuentra desde principios de febrero hasta finales de junio. Durante estos meses, se ha experimentado el primer contacto con el mundo laboral. Con estas prácticas se ha fomentado el trabajo en un entorno real. En el entorno real se trabaja con mucha gente con habilidades muy diversas entre ellos. Esto durante la carrera es más complicado de observar, ya que la mayoría de compañeros tienen una formación similar a la tuya. Por lo tanto, he aprendido a tratar con compañeros que tenían una formación distinta.
Por otro lado, este proyecto me ha servido para mejorar mi capacidad de resolución de problemas ya que, durante el desarrollo de este proyecto, me he encontrado con aspectos que no había trabajado nunca y he tenido que formarme sobre ellos.
Este proyecto ha sido muy interesante a nivel técnico ya que se han utilizado muchas tecnologías distintas. En el componente SINTRA se han utilizado tecnologías como *.NET* y *Angular* que se encuentran al orden del día en el desarrollo web. El componente *Integrador* está programado en Java ya que es un lenguaje altamente utilizado en el ámbito sanitario. Además, el framework *Spring Boot* ha sido una buena decisión dada su facilidad de uso y rapidez de desarrollo. Tambien, el uso de *Docker* para la creación del entorno de ejecución ha permitido llevar el mismo sobre las distintas máquinas de desarrollo y producción de una manera uniforme.

Apéndice A

Sistema SINTRA

El sistema SINTRA[1] es un sistema desarrollado por la empresa *Hiberus Tecnología* que tiene como objetivo la gestión integral de atención onmicanal de principio a fin. Es decir, este sistema plantea gestionar las relaciones entre los clientes y las empresas en ámbitos como la cita previa, la atención inteligente, la atención digital... Además, esta herramienta plantea también funcionalidades de análitica avanzada mediante cuadros de mando basados en la herramienta *PowerBI* de *Microsoft*[2]. En la Figura A.1, se muestran de forma conceptual las posibilidades de este sistema.
Actualmente (mayo 2021), al ser un proyecto nuevo (nacido durante la situación de confinamiento por la pandemia del COVID-19 en marzo de 2020), solo se encuentra implementado el módulo de *Cita previa*. Este módulo se encuentra desarrollado utilizando el *stack* tecnológico de *JavaScript* con el *framework Angular*, *C#* y *.NET Framework* y una base de datos en memoria para el desarrollo. No obstante, para la versión de producción se utiliza una base de datos *PostgreSQL*.

Este sistema de cita previa se encuentra actualmente implantado en el *Gobierno de Aragón*[3], el *Departamento de vivienda de San Sebastián*[4] y el sistema de cita previa de *DKV*[5] entre otras organizaciones.

Como se ha comentado anteriormente, la aplicación web de SINTRA está programada con el *framework Angular*. Con esta tecnología divide los módulos en distintos componentes. En la Figura A.2, se puede apreciar una captura de pantalla del módulo de reserva de una cita en el sistema SINTRA. En la Figura A.3, se puede apreciar la captura de pantalla del módulo de reserva de cita pero estando integrado con el sistema HIS. Como se puede observar, el sistema de cita previa pide los datos de contacto del usuario pero para realizar la integración (como se explica en el Capítulo 8) se deben añadir más campos a este formulario.

[1] `https://www.hiberus.com/sintra`
[2] `https://powerbi.microsoft.com/es-es/`
[3] `https://citaprevia.aragon.es/provincias`
[4] `https://www.euskadi.eus/web01-ejqmatic/es/zuzeneanwebbooking/`
[5] `https://citapreviasucursal.dkvseguros.com/appointment`

En el caso del lado del servidor, está programado utilizando *.NET Framework* con *C#*. La división de este sistema se realiza con la visión del modelo vista controlador (MVC). Los controladores de SINTRA son los encargados de dar soporte a las llamadas de la aplicación web mediante el acceso vía *API REST*.

Figura A.1: Diagrama conceptual de SINTRA

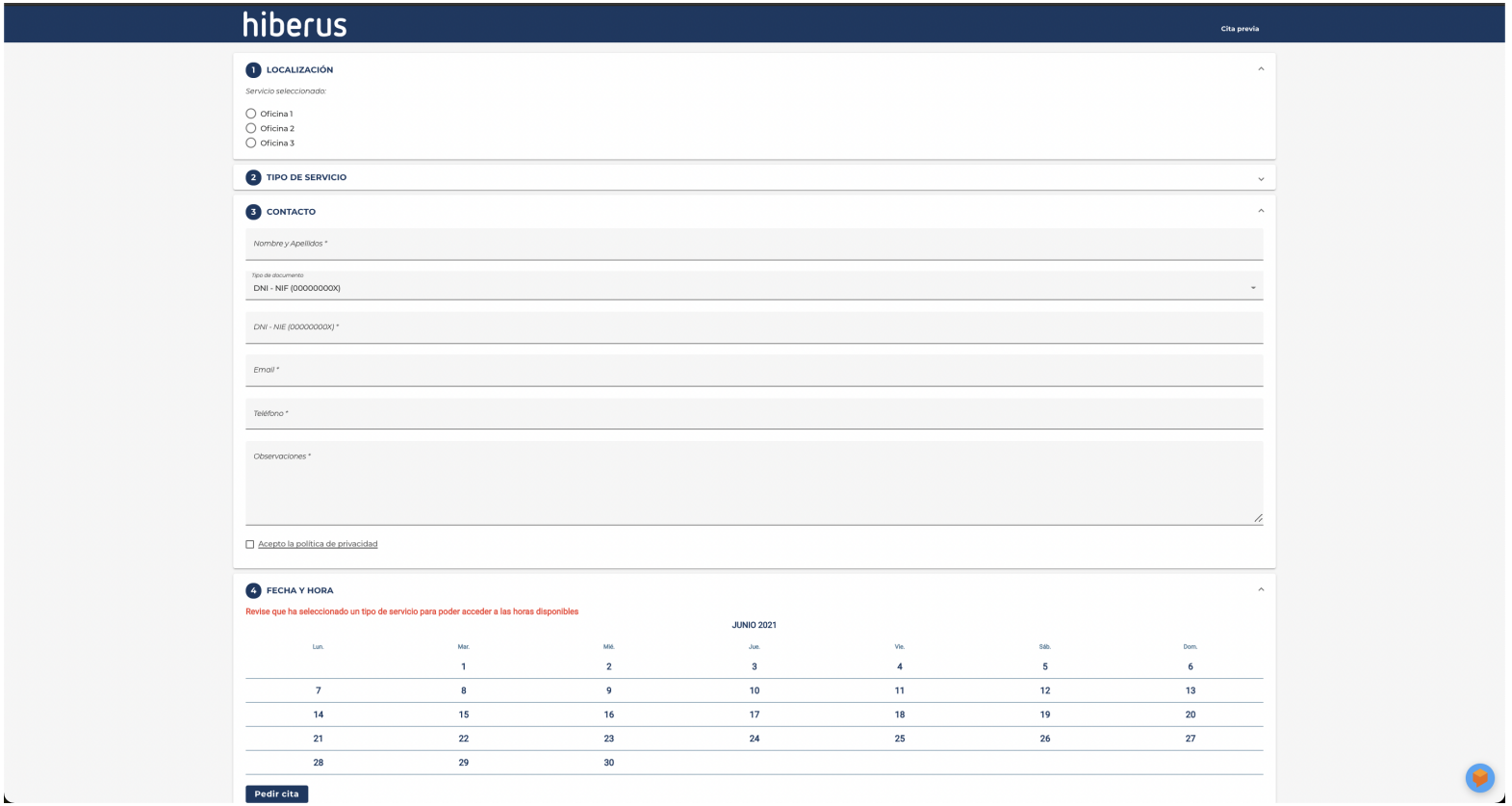

Figura A.2: Pantalla de reserva de cita previa sin integración con HIS

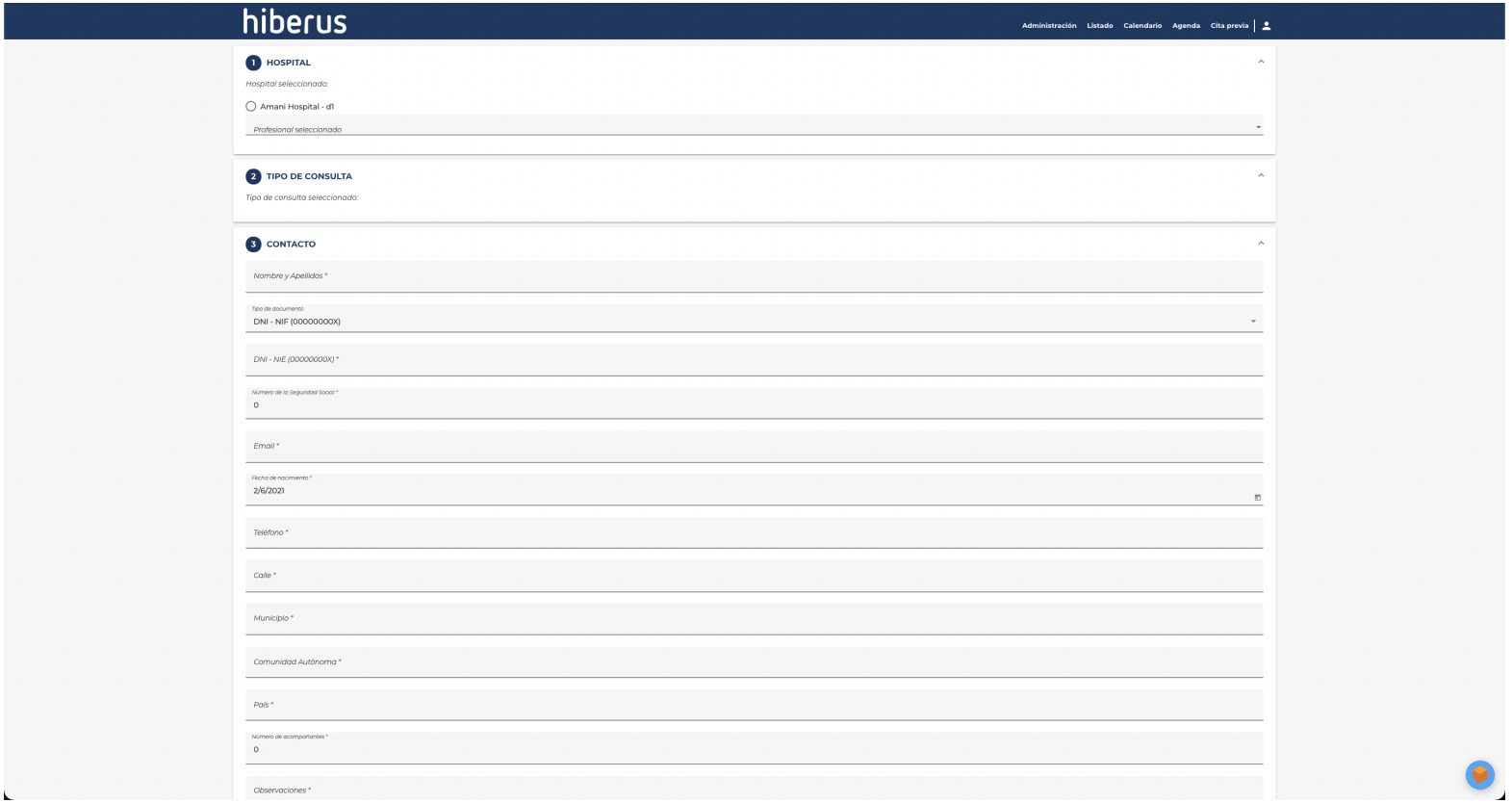

Figura A.3: Pantalla de reserva de cita previa con la integración con HIS

Apéndice B

Comparación de sistemas HIS de código abierto

Se ha optado por soluciones de código abierto (*Open Source*). Se han evaluado las distintas opciones, poniendo especial énfasis en los aspectos relevantes con la integración con sistemas externos.

B.1. Bahmni

Se trata de un HIS[1] con licencia *AGPL 3*, basado en sistemas *Open Source* como *Odoo* (ERP, Enterprise Resource Planning), *OpenMRS* (HIS) y *OpenELIS* (LIS, Laboratory Information System). Estos sistemas llevan a cabo correctamente sus tareas y permiten una gestión integral de un hospital. Los puntos a favor de este *HIS* se centran en que su desarrollo está muy activo y se basa en proyectos grandes. Los puntos en contra, en cambio, destacan la instalación es complicada ya que debe ser sobre el sistema operativo *CentOS* aunque podría ser con *Docker* pero aumentaría la complejidad. Además, este sistema cuenta con muchas capas más de las necesarias, ya que no interesa los aspectos de gestión de inventario ni de laboratorio. Estas capas extra, por lo tanto, aumentan la complejidad del sistema.

B.2. OpenMRS

Se trata del sistema HIS *Open Source* de referencia[2]. Se distribuye bajo una licencia *Mozilla Public License 2.0*. Los puntos a favor de este sistema son: la facilidad de instalación con *Docker*, su desarrollo activo, su interfaz de acceso vía *API REST* [13], su amplia documentación y su interfaz FHIR [7]. El aspecto en contra de este sistema es la falta de implementación de los métodos necesarios para soportar el estándar HL7 [1] y por lo tanto habría que buscar un método de integración alternativo. Este método pasaría por integrar usando FHIR o

[1] `https://www.bahmni.org/`
[2] `https://openmrs.org/`

un componente externo como un bus de integración para realizar la integración directamente contra la base de datos del HIS.

B.3. OpenHospital

Este sistema HIS se trata de un sistema *Open Source* distribuido bajo una licencia *GPL 3*[3]. Un punto a favor de este sistema es que es muy ligero, pudiendo ser distribuido únicamente como un fichero *JAR* sobre la máquina. Los puntos en contra son: que la base de datos no se distribuye junto con el fichero ejecutable y, además, no cuenta con soporte para mensajería HL7.

B.4. GNUHealth

El HIS desarrollado por la asociación *GNU Solidario* se distribuye bajo una licencia *GPL 3*[4]. Los puntos a favor de este sistema se basan en que el proyecto cuenta con un desarrollo muy activo y, además, posee un módulo desarrollado de forma independiente para soportar el estándar FHIR. No obstante, como punto en contra, no soporta mensajería HL7.

B.5. OpenEMR

Este sistema HIS se distribuye bajo una licencia *AGPL 3*[5]. Los puntos a favor para este sistema son su instalación, que con *Docker* es sencilla y su desarrollo activo. También, es importante destacar que cuenta con una interfaz REST y permite usar el estándar FHIR, aunque no está todo soportado. Como punto en contra, se puede destacar que no cuenta con una interfaz HL7 para comunicarse y por lo tanto habría que utilizar un bus de integración para llevar a cabo la misma.

B.6. Conclusión

Por lo tanto, comparando los puntos anteriores, se ha elegido *OpenMRS* como HIS a utilizar. Para este proyecto interesa que el HIS sea capaz de procesar mensajes HL7 y peticiones FHIR. El HIS elegido no tiene la capacidad de procesar todos los tipos de mensajes HL7. Por lo tanto, se deberá realizar la integración de los mensajes no soportados utilizando un bus de integración. Con este componente se podrán usar distintas estrategias. La estrategia de integración más directa es el uso de la base de datos del sistema HIS.

[3] `https://www.open-hospital.org/en/`
[4] `https://www.gnuhealth.org/`
[5] `https://www.open-emr.org/`

Apéndice C

Pruebas y validación

En este capítulo se van a detallar las pruebas realizadas en el proyecto. Como se comenta en el Capítulo 4 se han realizado tanto *tests* unitarios como de integración. En la Sección C.1 se describen las pruebas realizadas para la validación del módulo de horarios. En la Sección C.2 se trata la validación del módulo de citas. En la Sección C.3 se describen las pruebas realizadas para la validación del módulo de pases de acceso. Por último, en la Sección C.4 se muestra la validación realizada sobre la integración de SINTRA con el componente *Integrador*.

C.1. Validación del módulo de horarios

Para la realización de pruebas sobre el módulo de horarios se han realizado *tests* de integración para comprobar que se cumplía la especificación de la interfaz (Sección 5.1). Se han probado los casos de funcionamiento correcto y los casos de que el parámetro de la fecha no tenga el formato específico.
Además, para comprobar el funcionamiento de las funciones auxiliares necesarias (como la de validación del formato de la fecha) se han utilizado *tests* unitarios. En estos *tests* se comprueban los casos límite para ver si se cumple la especificación de estas funciones. Las funciones sobre las que se han realizado *tests* unitarios son: `Validators.validateFecha(string)`, `Utils.splitName(string)` y sobre la clase `HorariosService`.
La función `Validators.validateFecha(string)`, devuelve un valor verdadero si la fecha que se pasa como parámetro tiene el formato *yyyy-MM-dd*. Las pruebas sobre esta función se basan en dos comprobaciones. La primera es acerca de si la fecha cumple el formato con una fecha con ese formato. La segunda comprobación se basa en comprobar que una fecha que no cumple el formato no es aceptada por la función.
La función `Utils.splitName(string)`, devuelve una tupla (lista de longitud 2) que separa la *string* del nombre del usuario en la primera posición y su apellido en la segunda posición. Para probar esta función se han utilizado 3 casos de pruebas. El primero es el caso de aceptación, es decir, con un nombre válido devuelve la lista con el nombre en la primera posición y su apellido en la segunda. El segundo caso comprueba el correcto funcionamiento si el nombre es compuesto (por ejemplo: *José Luis Pérez*). El tercer caso comprueba el funcio-

namiento en caso de que la cadena que se pasa como parámetro a la función sea vacía (“”).
La clase *HorariosService*, se trata de una clase cuya funcionalidad se encuentra en torno a los horarios. Esta clase envía los mensajes de consulta de horarios al bus de integración y además se encarga de la funcionalidad de tratar el mensaje devuelto para obtener la lista de los horarios disponibles. La prueba realizada para comprobar que la clase envía y recibe información vía *HTTP* se basa en falsear el envío y recepción de un mensaje. Tras el falso envío se comprueba que el mensaje recibido tiene el valor que debería.
La función de tratamiento de casos de prueba tiene como parámetro el mensaje de respuesta (*SQR_S25*) y devuelve una lista de *string* que contienen los horarios en formato HH:mm. Para comprobar el correcto funcionamiento se han comprobado dos casos. En ambos casos se comprueba que la longitud de la lista devuelta coincide con el campo *QAK.4* que contiene la información sobre el número de resultados devueltos. En el primer caso se comprueba con un mensaje vacío, es decir, el mensaje no contiene horarios y el *QAK.4* es 0. En el segundo caso se comprueba con un mensaje que contiene 13 horarios, es decir, el *QAK.4* tiene valor 13.

A modo de resumen, se han sintetizado todas las pruebas realizadas en la Tabla C.1. En esta tabla aparece la función sobre la que se ha realizado la validación junto con los casos probados.

Función	**Prueba**
`validateFecha(string)`	La string cumple el formato HH:mm
`validateFecha(string)`	La string no cumple el formato.
`splitName(string)`	La string cumple el formato.
`splitName(string)`	La string cumple el formato con un nombre compuesto.
`splitName(string)`	La string no cumple el formato. Cadena vacía.
`queryTimeslots(string)`	El mensaje que se devuelve es el esperado por el test.
`getHorariosFromMessage(message)`	El mensaje contiene horarios.
`getHorariosFromMessage(message)`	El mensaje no contiene horarios.
`Controlador de consulta de horarios`	Test de integración para comprobar el correcto funcionamiento.
`Controlador de consulta de horarios`	Test de integración con la fecha que no cumple el formato (yyyy-MM-dd).

Tabla C.1: Pruebas realizadas para la validación del módulo de consulta de horarios

C.2. Validación del módulo de citas

Para la validación del módulo de citas se han creado *tests* unitarios y de integración para cubrir los distintos casos que se pueden dar según la especifi-

cación. Se han creado 11 tests de integración para probar los casos de la API desarrollada. Para la reserva de citas se han probado los casos de funcionamiento correcto, de fecha con un formato incorrecto, de la hora de la reserva con un formato incorrecto y, por último, el caso de que el paciente no existe. Para la modificación de una cita se ha probado el caso correcto, el caso de que el identificador de la cita no exista, el caso de que el paciente no exista, el caso de que el paciente de la modificación no coincida con el de la reserva, el caso de que el servicio de la modificación no coincida con el de la reserva y el caso de intentar modificar la cita a un hueco que ya está reservado por otra cita. Para la cancelación de citas se han probado dos casos: el caso de que la cita a modificar exista y el caso de que la cita a modificar no existe.
Se han realizado *tests* unitarios sobre las funciones: `Validators.validateHora(string)`, `PacientesService.getPatientData(DatosPersonalesUsuario)` y `Utils.getFHIRPatientIdentifier(Patient)`.
La función `Validators.validateHora(string)`, devuelve verdadero si la hora que se pasa como parámetro está en formato HH:mm, es decir $HH \in [0, 23]$ y $mm \in [0, 59]$. Se han comprobado los casos de: funcionamiento correcto, hora incorrecta, minutos incorrectos, hora y minutos incorrectos y el caso de separador incorrecto.
La función `PacientesService.getPatientData(DatosPersonalesUsuario)`, se utiliza para realizar la petición a la interfaz FHIR del sistema HIS. Esta función devuelve el registro FHIR del usuario cuya información coincide con el parámetro pasado. Además, si no se encuentra un usuario que cumpla con los requisitos se lanza una excepción `UserNotFoundException`. Se han comprobado los casos de que el paciente exista en el sistema HIS y el caso de que no exista.
Por último, la función `Utils.getFHIRPatientIdentifier(Patient)`, se encarga de devolver un identificador de paciente (*UUID*) dado su registro FHIR. El identificador FHIR se trata de una URI identificativa para el paciente pero en este caso interesa obtener una cadena de caracteres que contenga solo su identificador. Para probar este comportamiento se han probado los casos de: funcionamiento correcto, identificador sin *UUID* (es decir, URI incompleta) y el caso en el que la URI no tenía el formato válido (por ejemplo, “”).

Como resumen de esta sección se ha creado la Tabla C.2. Esta tabla contiene las funciones sobre las que se han realizado las pruebas correspondientes a la validación del módulo de citas.

C.3. Validación del módulo de pases de acceso

Para la validación del módulo de pases de acceso se han creado *tests* unitarios y de integración. Los *tests* de integración se han creado para la verificación del funcionamiento de la API creada. Para comprobar el método de obtener la información de un pase de acceso se prueban los casos de: obtener un pase que existe y obtener un pase que no existe. Para probar el método de obtener el código QR asociado a un pase de acceso se comprueban los casos de: obtener el código QR para un pase que existe y obtener el código QR para un pase que no existe. Para el método de validar y entrar con un pase de acceso se comprueban

los casos: de validar un pase correcto (es decir, el pase de acceso existe y tiene usos restantes), el caso de que el pase de acceso no existe, el caso de que el pase de acceso no tiene usos restantes (0) y el caso en que los usos restantes son negativos.
Se han creado *tests* unitarios para la clase `PaseAccesoService`. Esta clase es la encargada de realizar operaciones sobre los pases de acceso. Se han creado pruebas para la función de crear el código QR, en la cual se comprueba que dado un pase de acceso se ha creado correctamente el código QR y para la función de crear un pase de acceso, en la cual dado un objeto de la `PeticionCita` se crea un pase de acceso.

Para resumir las pruebas realizadas sobre el módulo de pases de acceso, se puede observar la Tabla C.3. Esta tabla contiene una descripción sobre las pruebas realizadas a las funciones de este módulo.

C.4. Validación de la integración con SINTRA

La validación del sistema SINTRA, como se comenta en el Capítulo 8, ha sido manual. Esta decisión está basada en que el sistema no cuenta con *tests* automatizados y todas las pruebas se realizan manualmente. Por lo tanto, para la validación de este sistema se ha utilizado al cliente (la empresa *Hiberus*) como usuarios para validar el funcionamiento y de esta forma obtener una retroalimentación más detallada que mediante el uso de pruebas automatizadas.

Función	**Prueba**
`validateHora(string)`	La string cumple el formato HH:mm
`validateHora(string)`	La string no cumple el formato. Probado con: Hora incorrecta, minutos incorrectos, hora y minutos incorrectos y separador incorrecto.
`getPatientData( DatosPersonalesUsuario)`	Test de integración que comprueba la existencia de un paciente que existe.
`getPatientData( DatosPersonalesUsuario)`	Test de integración que comprueba la existencia de un paciente que no existe.
`getFHIRPatientIdentifier (Patient)`	Prueba con una URI con el formato correcto.
`getFHIRPatientIdentifier (Patient)`	Prueba con una URI con el UUID vacío.
`getFHIRPatientIdentifier (Patient)`	Prueba con una URI vacía (“”).
`Controlador de reserva`	Test de integración para comprobar el correcto funcionamiento de la reserva de citas.
`Controlador de reserva`	Test de integración para la reserva de citas con la fecha mal formateada.
`Controlador de reserva`	Test de integración para la reserva de citas con la hora mal formateada.
`Controlador de reserva`	Test de integración para la reserva de citas con un paciente que no existe.
`Controlador de modificación`	Test de integración para comprobar el correcto funcionamiento de la modificación de citas.
`Controlador de modificación`	Test de integración para la modificación de citas con un identificador de cita que no existe.
`Controlador de modificación`	Test de integración para la modificación de citas con un paciente que no existe.
`Controlador de modificación`	Test de integración para la modificación de citas con un paciente que no coincide con el que ha reservado la cita.
`Controlador de modificación`	Test de integración para la modificación de citas con un servicio que no coincide con el que la cita ha sido reservada.
`Controlador de modificación`	Test de integración para la modificación de citas con un hueco que está reservado por otra cita.
`Controlador de cancelación`	Test de integración para comprobar el correcto funcionamiento de la cancelación de citas.
`Controlador de cancelación`	Test de integración para la cancelación de citas con un identificador de cita que no existe.

Tabla C.2: Pruebas realizadas para la validación del módulo de citas

Función	**Prueba**
`createQrCode()`	Comprueba el correcto funcionamiento de la creación del código QR dado un pase de acceso.
`createPaseAcceso()`	Comprueba el correcto funcionamiento de la creación de un pase de acceso dada una petición de una cita.
`Controlador de obtener pase`	Test de integración que comprueba el correcto funcionamiento de obtener un pase de acceso existente.
`Controlador de obtener pase`	Test de integración para obtener la información de un pase de acceso no existente.
`Controlador de obtener QR`	Test de integración que comprueba el correcto funcionamiento de obtener el código QR de un pase de acceso existente.
`Controlador de obtener QR`	Test de integración para obtener el código QR de un pase de acceso no existente.
`Controlador de validar pase`	Test de integración que comprueba el correcto funcionamiento del método de verificación de un pase de acceso existente.
`Controlador de validar pase`	Test de integración para validar la información de un pase de acceso no existente.
`Controlador de validar pase`	Test de integración para validar la información de un pase de acceso existente pero sin usos restantes (usos = 0).
`Controlador de validar pase`	Test de integración para validar la información de un pase de acceso existente y con el número de usos con valor negativo (usos $\leq$ 0).

Tabla C.3: Pruebas realizadas para la validación del módulo de pases de acceso

Apéndice D

Funcionamiento del bus de integración

El sistema *Mirth* necesita instalar un programa adicional (*Mirth Connect Administrator Launcher*[1]). Este programa se conecta vía *HTTPS* al bus de integración y, usando las credenciales especificadas en el *Docker* se inicia sesión como se muestra en la Figura D.1.

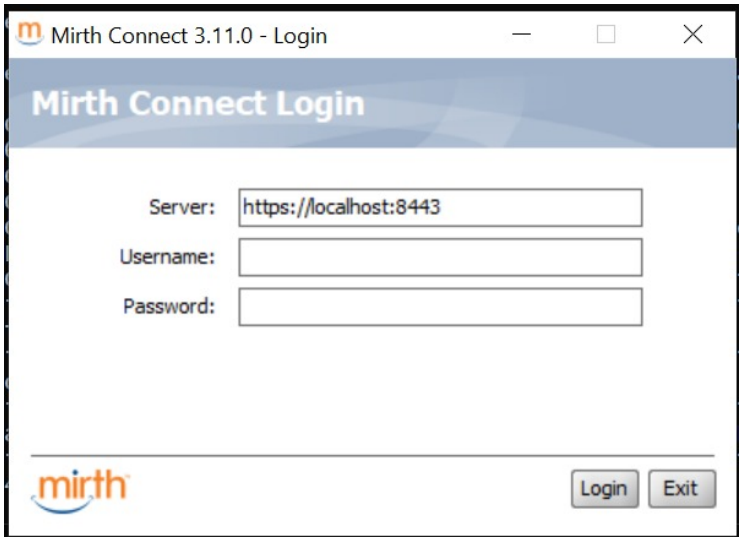

Figura D.1: Página de inicio de sesión en la herramienta *Mirth Connect Administrator*

Una vez se ha iniciado sesión, se tiene acceso a un cuadro de mando (*dashboard*) que muestra información acerca de los canales del sistema.
En *Mirth* se utilizan canales para escuchar, transformar, y realizar acciones mediante los mensajes HL7. Los conectores para recibir mensajes son muy variados, puede ser mediante *HTTP*, mediante *Web Services* (*SOAP*), mediante

[1] `https://www.nextgen.com/products-and-services/nextgen-connect-integration-engine-downloads`

TCP, mediante encuesta de una fuente de datos (base de datos, ficheros) cada cierto tiempo, mediante la escucha activa de una cola de mensajes *JMS*. De igual manera, para enviar los mensajes se pueden usar distintos tipos de conectores de destino.
Los transformadores son pasos de modificación del mensaje, consulta de fuentes de datos y asignación de campos del mensaje a variables del canal. Estos componentes cuentan con plantillas de mensajes de entrada y salida de tal forma que se puede asignar una variable a un campo del mensaje de salida.
El flujo de procesamiento de los mensajes de *Mirth* se divide en 10 pasos [14]:

1. El conector de origen recibe el mensaje o un flujo de mensajes.
2. Si los lotes están activados en el conector de origen se dividen los mensajes/flujos en mensajes individuales. A partir de este paso se estará trabajando con mensajes individuales.
3. El mensaje pasa por el *preprocesador*.
4. El mensaje pasa por el filtro de origen. Este filtro determina si el mensaje puede ser procesador por el canal o no.
5. El mensaje pasa por el transformador de origen.
6. El mensaje pasa por cada destino especificado de forma secuencial. En primera instancia se evalúa el filtro del destino para comprobar que ese mensaje puede ser procesado por ese destino. En caso de que no sea así, se pasa al siguiente destino de la lista. Si el mensaje puede ser procesado pasa al siguiente paso.
7. El mensaje pasa por el transformador del destino.
8. El mensaje pasa por el conector de destino. Este conector ejecutará las acciones necesarias para escribir o enviar el mensaje de acuerdo a lo programado. Cuando este paso termina, se vuelve al paso 6 para evaluar el siguiente destino de la lista.
9. Después de que se ejecuten todos los destinos se ejecuta el *postprocesador*. El mensaje no puede ser modificado por el usuario a partir de este punto.
10. Después de que se ejecute el *postprocesador* se tiene que contestar al sistema que ha enviado el mensaje (en caso de que sea necesario). Si el conector de origen lo permite, existe una opción de generar una respuesta *ACK*. En caso contrario, se puede responder utilizando cualquier variable del diccionario de respuestas o una respuesta personalizada de un destino que puede ser modificada mediante los transformadores de la respuesta del destino.

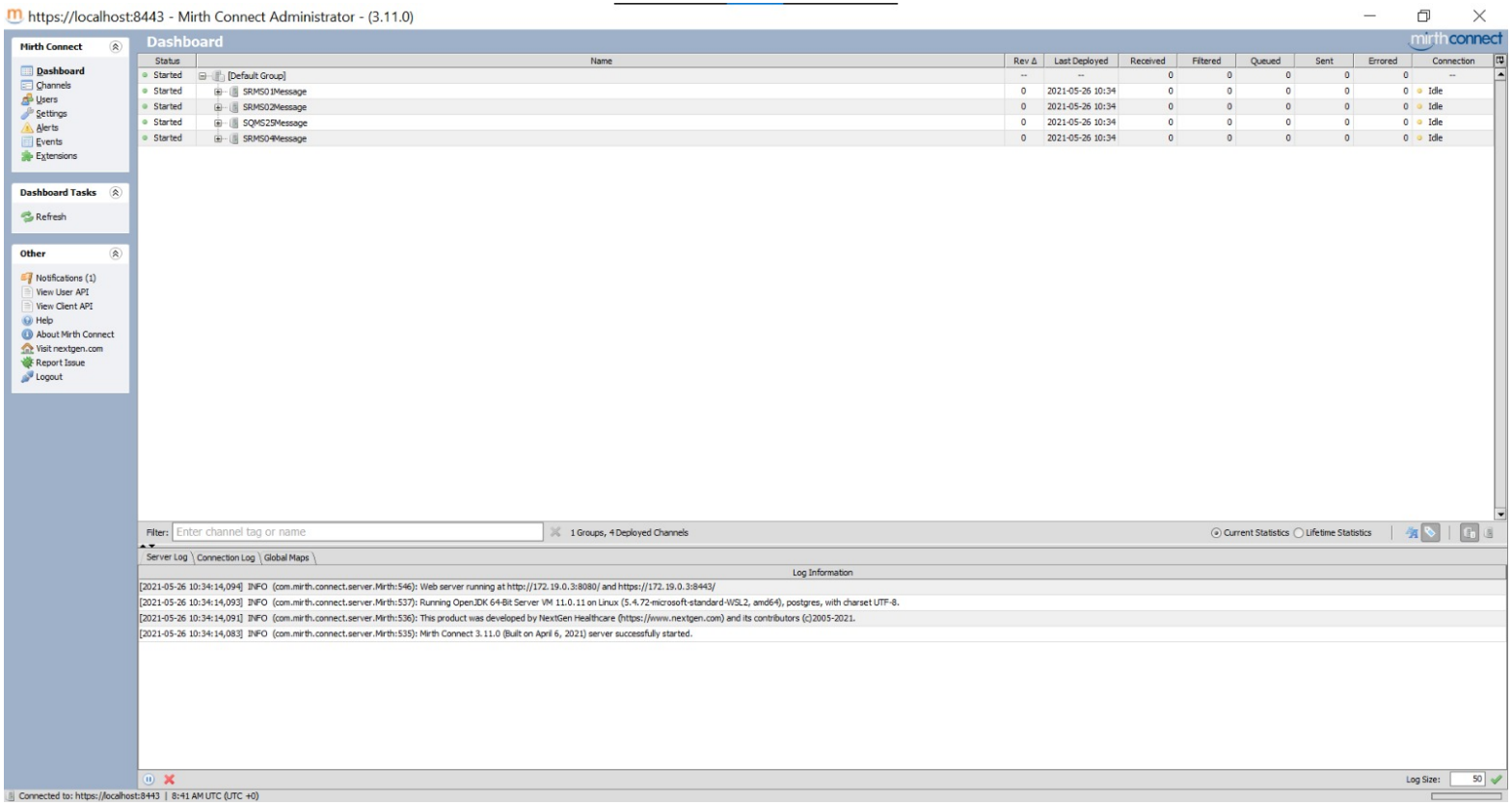

Figura D.2: Página de *dashboard* de la herramienta *Mirth Connect Administrator*

Apéndice E

Diagramas de interacción

En este anexo se van a recopilar los diagramas de interacción de los distintos casos de uso. La Figura E.1 muestra el diagrama de secuencia del caso de uso de obtener los horarios disponibles para una fecha dada. La Figura E.2, contiene el diagrama de secuencia del caso de uso de reserva de una cita en el sistema sanitario. En la Figura E.3, se puede observar el diagrama de modificación de una cita ya existente. La Figura E.4, muestra el diagrama correspondiente al caso de uso de cancelación de una cita existente. El caso de uso de obtener los datos del paciente se corresponde con la Figura E.5. En la Figura E.6, muestra el diagrama que describe la creación de un pase de acceso asociado a una cita del sistema sanitario. En la Figura E.7, se puede observar el caso de uso de acceder al centro mediante un pase de acceso.

E.1. Consulta de horarios

En la Figura E.1 se puede observar la comunicación entre los distintos componentes para el caso de uso de la consulta de horarios disponibles. Para este caso de uso se realiza una llamada al sistema SINTRA para obtener los horarios dado un profesional, un servicio y una fecha. SINTRA consulta el sistema HIS por medio del componente *Integrador*. El componente *Integrador* crea el mensaje HL7 necesario y lo envía al bus de integración que se encargará de obtener la información necesaria con la estrategia especificada.

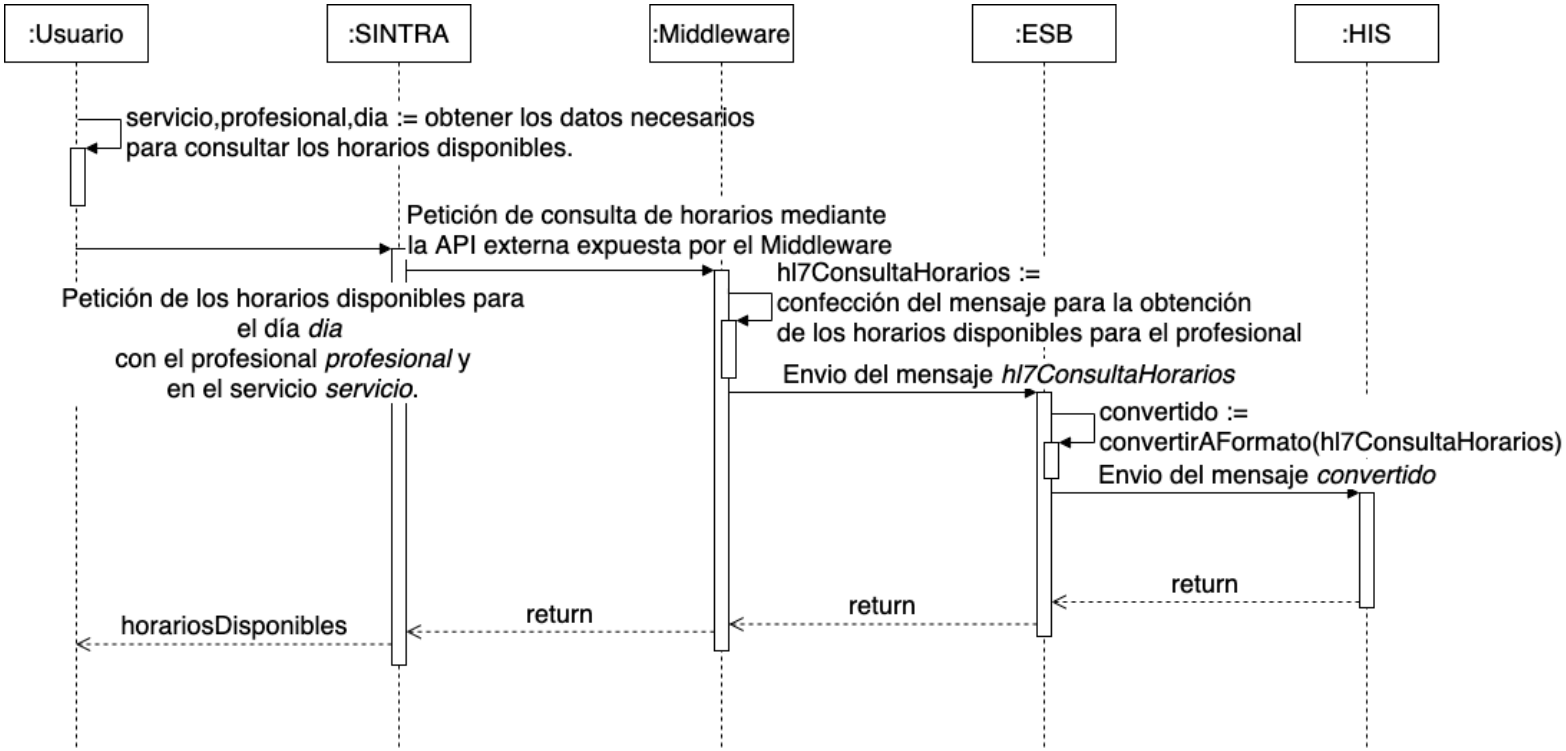

Figura E.1: Diagrama de secuencia de obtener los horarios disponibles

E.2. Reserva de cita

Para la reserva de una cita se utiliza el diagrama de la Figura E.2. Para reservar una cita se tendrá que haber utilizado el caso de uso de obtener los horarios disponibles, ya que la cita debe ser reservada en un hueco libre. En este diagrama se puede observar que una vez realizada la petición de reserva de cita en SINTRA se consulta al HIS por medio del componente *Integrador*. Este componente utiliza el caso de uso de obtener la información del paciente (Figura E.5) y, en caso de que el paciente no exista, se devolverá un error al sistema SINTRA. Si la cita se ha reservado correctamente en el sistema HIS, se devuelve un mensaje de éxito a SINTRA y almacena la cita en su base de datos.

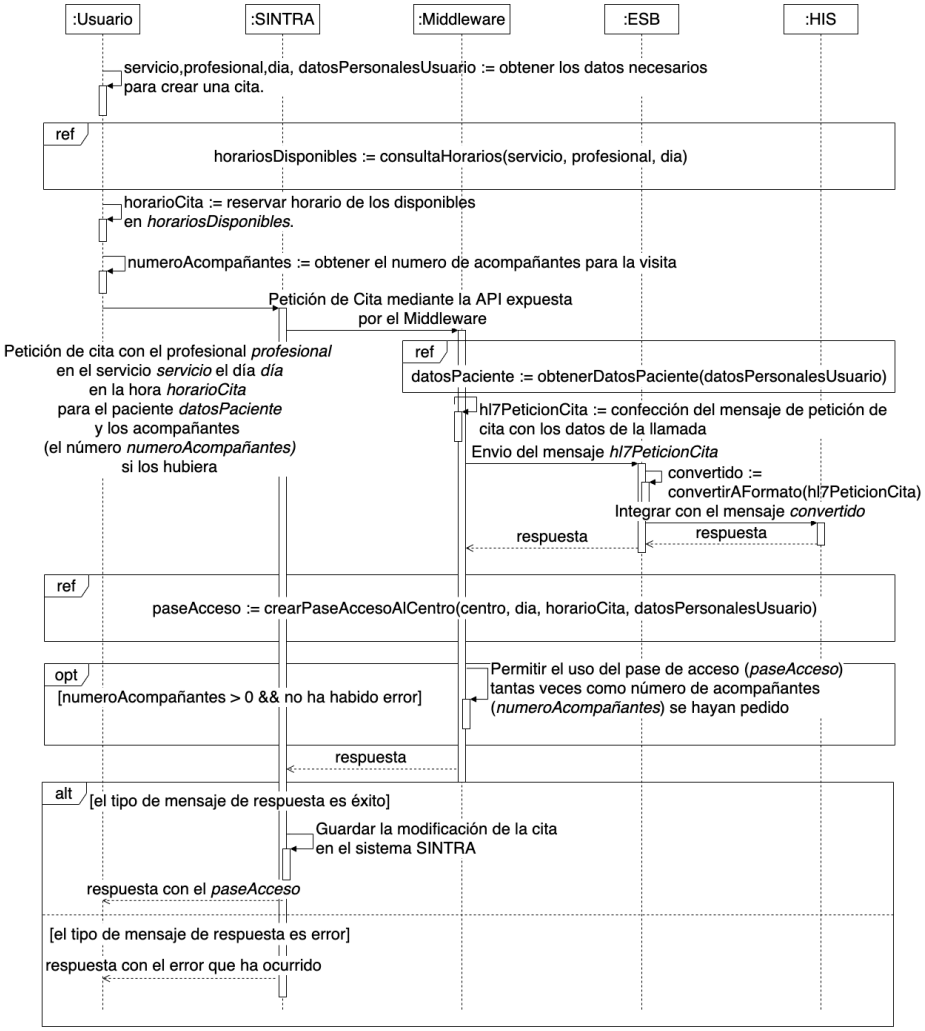

Figura E.2: Diagrama de secuencia de reserva de una cita

E.3. Modificar cita

Para modificar una cita existente, se sigue el diagrama que se refleja en la Figura E.3. El trabajador obtiene la información almacenada en SINTRA acerca de la cita a modificar, realiza las modificaciones necesarias y guarda la cita. Al guardar la cita en el sistema SINTRA se produce una llamada al componente *Integrador* que envía la información modificada al sistema HIS. Además, en esta petición también se comprueba la identidad del paciente con el caso de uso de la Figura E.5. En caso de que en el sistema HIS se modifique su cita correctamente, se actualizará la información en el sistema SINTRA.

E.4. Cancelar cita

En la Figura E.4 se muestra el flujo de cancelación de una cita existente. Como ocurre en el caso de uso de modificar una cita existente (Figura E.3), el primer paso es obtener la información de la cita creada. Tras ello, se envía la petición de cancelación al sistema SINTRA. Después, el sistema SINTRA hace una llamada al componente *Integrador*, que se comunicará por mensajería HL7 con el bus de integración y el sistema HIS. Si la cita se ha eliminado correctamente, se eliminará el pase de acceso asociado a ella y se devolverá un mensaje de éxito al sistema SINTRA que eliminará la cita de su base de datos.

E.5. Obtener los datos de un paciente

Para la obtención de datos de un paciente, se sigue el diagrama de la Figura E.5. Esta petición se realiza entre el componente *Integrador* y el sistema HIS mediante la interfaz FHIR del mismo.

E.6. Crear un pase de acceso

La creación de los pases de acceso se rige por el diagrama de la Figura E.6. Este pase de acceso se crea con la información de la cita. Para obtener más información acerca de la implementación de los pases de acceso se puede consultar el Capítulo 7.

E.7. Verificar un pase de acceso

La Figura E.7 refleja la información acerca de la verificación de un pase de acceso. En este diagrama se comprueba que el pase de acceso tenga fecha actual y además que tenga usos restantes. Más información acerca de la implementación de los pases de acceso se puede encontrar en el Capítulo 7.

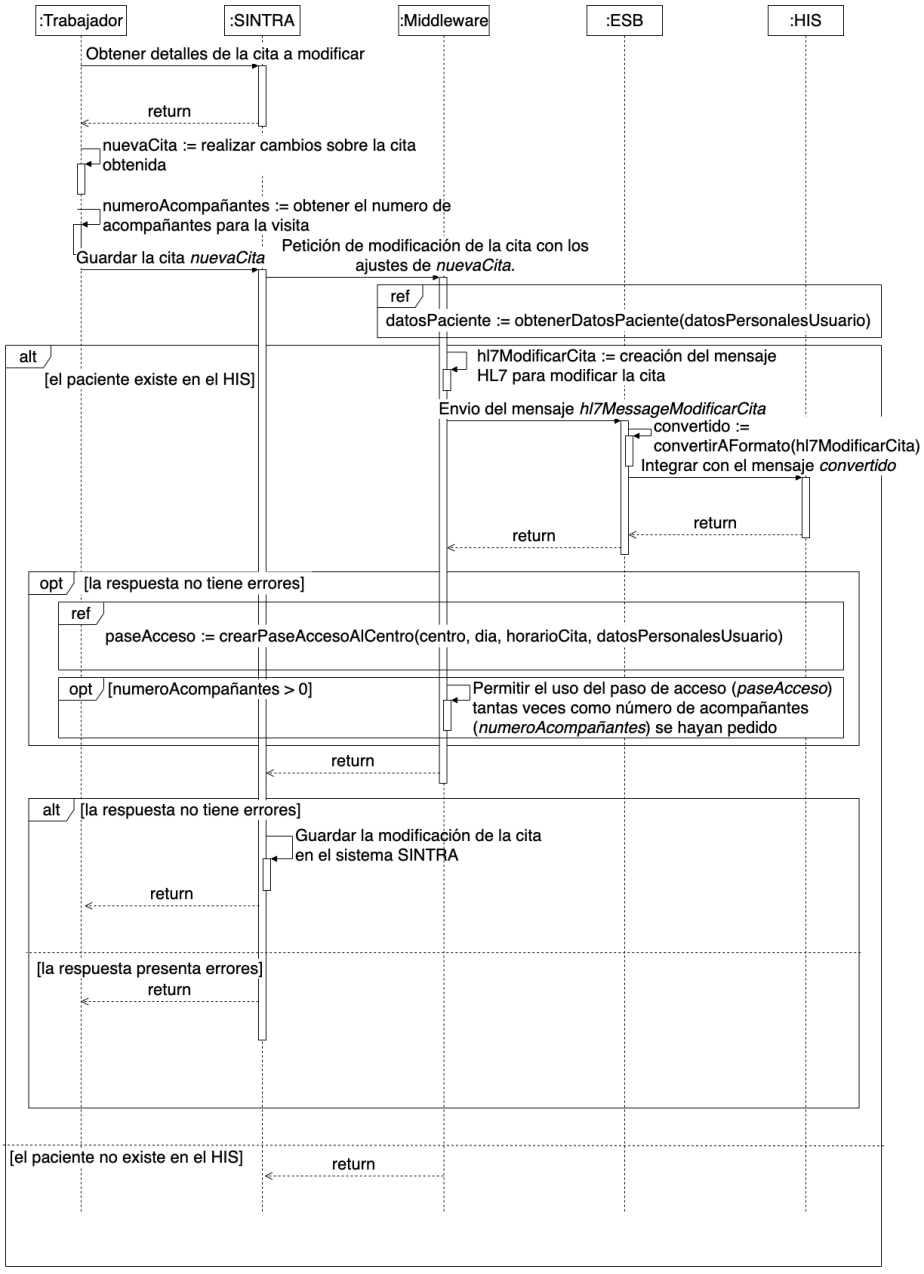

Figura E.3: Diagrama de secuencia de modificación de una cita existente

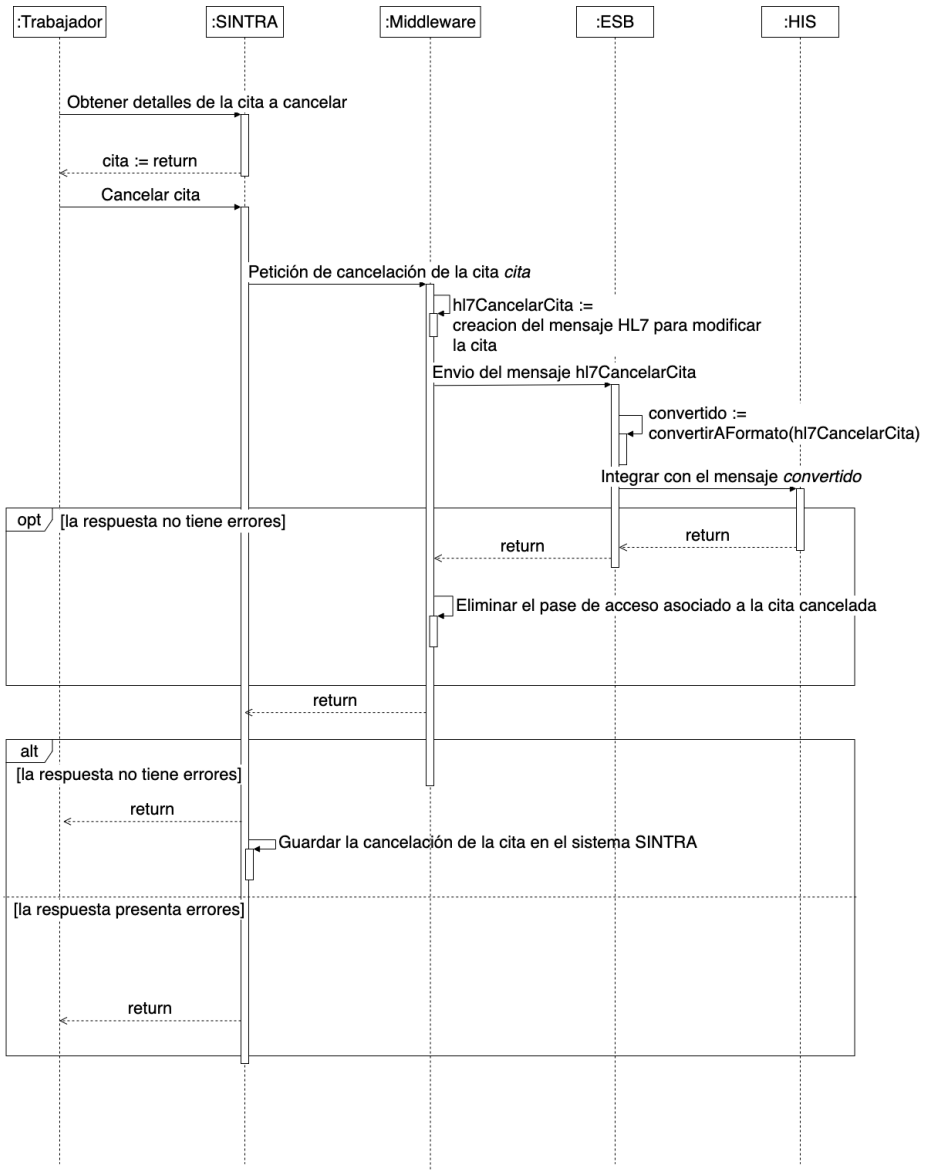

Figura E.4: Diagrama de secuencia de cancelar una cita existente

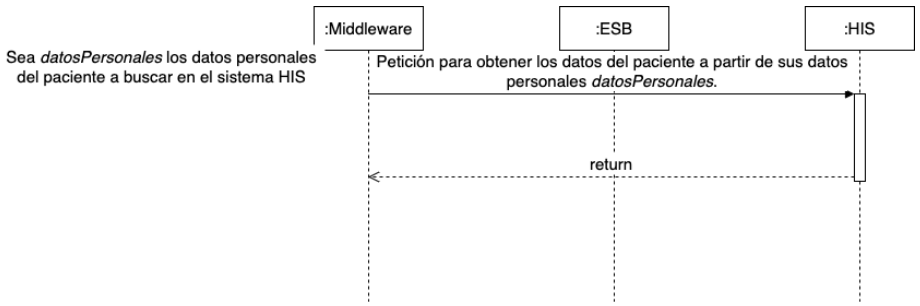

Figura E.5: Diagrama de secuencia de obtener los datos de un paciente del sistema HIS

Figura E.6: Diagrama de secuencia de creación de un pase de acceso

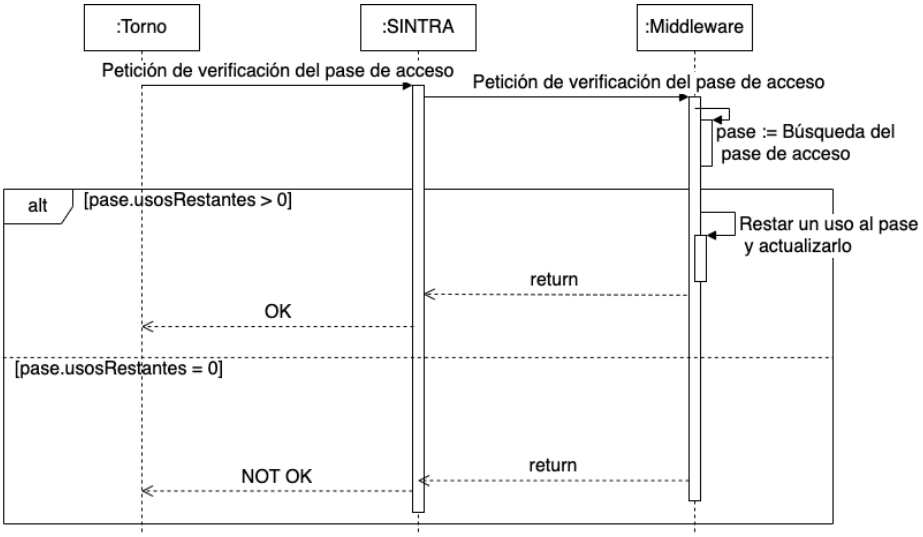

Figura E.7: Diagrama de secuencia de verificación de un pase de acceso

Apéndice F

Estructura y contenidos de los mensajes HL7

La estructura de los mensajes HL7 utilizados se comentará en este apéndice. La estructura de estos mensajes está basada en la utilizada por la *Gerencia Regional de Salud de la Junta de Castilla y León* [15] y cumpliendo el estándar según se muestra en la página de *Caristix*[1]. Los mensajes utilizados en este proyecto se corresponden con los de consulta de horarios disponibles *SQM_S25* y su respuesta (Sección F.1). Para la creación de una cita, se utilizará el mensaje *SRM_S01* y su respuesta (Sección F.2). Para la modificación de una cita, se usarán los mensajes *SRM_S02* y su respuesta (Sección F.3). Por último, para la cancelación de una cita se usarán los mensajes *SRM_S04* y su respuesta, en la Sección F.4.

[1] `https://hl7-definition.caristix.com/v2/HL7v2.5`

F.1. SQM_S25 y SQR_S25

Estos mensajes se utilizan para la consulta de horarios disponibles para un profesional dado en una fecha determinada. Su estructura es la siguiente:

```
MSH|^~\&|LOCAL|LOCAL|Their System|Their Remote Facility|<
    date>||SQM^S25^SQM_S25|<id>|P|2.5
QRD|<query_datetime>|R|I|<query_id>||||<profesional_name
    >|SBK|<location_name>
QRF|<service_type>||||||||^^^<start_date>^<end_date>
```

La estructura de este mensaje es el de una consulta en la cual se envían los datos necesarios para realizar la misma. Se puede observar que se envían los datos de nombre del profesional, localización, tipo de servicio (consulta) y las fechas iniciales y finales para buscar los huecos disponibles.
Para el mensaje de respuesta se utiliza la siguiente estructura:

```
MSH|^~\&|Their System|Their System|LOCAL|LOCAL|<date>||
    SQR^S25^SQR_S25|<id>|P|2.5
MSA|<ack_code>|<answers_to>|<error_reason>?
QAK||||<result's_length>
[
    SCH||||||timeslotQuery||||||||||integradorQuery||||
        integradorQuery
    TQ1|<id>||||||<slot_start_date>|<slot_end_date>
    RGS|<id>
]{0,n}
```

Este mensaje se crea como respuesta al anterior. El segmento *MSA* contiene información acerca de los huecos disponibles de los horarios solicitados. Además como es posible que se devuelva más de un hueco para el día solicitado el segmento *QAK* contiene un campo que indica el número de resultados que devuelve. Para cada resultado devuelto se replican los segmentos *SCH*, *TQ1* y *RGS* aunque este último no es obligatorio. Como se puede observar, los valores del segmento *SCH* son fijos y solo indican quién ha realizado la acción. En el segmento *TQ1* se encuentra la información acerca de cuando empieza y termina el hueco disponible. Los identificadores que se encuentran en estos segmentos repetidos hacen referencia a su posición en el mensaje, es decir, toman valor desde *1* hasta *n*.

F.2. SRM_S01 y SRR_S01

Esta pareja de mensajes se utilizan para la reserva de citas en el sistema HIS. El mensaje de reserva *SRM_S01* tiene la siguiente estructura:

```
MSH|^~\&|LOCAL|LOCAL|Their System|Their Remote Facility|<
    date>||SRM^S01^SRM_S01|<id>|P|2.5
ARQ|<appointment_uuid>||||||<appointment_reason>||||<
    appointment_start_date>||||<profesional>||||^^
    integrador
PID|||^^^^<patient_uuid>||<patient_name>||||||<
    patient_address>
RGS|1
AIS|1|A|<service_id>^<service_name>
AIL|1|A|^^^&<service_location>
AIP|1|A|^<profesional>
```

En el mensaje anterior se puede observar que en el segmento *ARQ* se envía un *appointment_uuid*. Este será el que se almacene en la base de datos del HIS y con el que se pueda hacer referencia a la cita desde los sistemas SINTRA e *Integrador*. Además, en el segmento *AIS* se especifica el servicio para la cita y se hace mediante su identificador numérico y su nombre. El profesional encargado de la cita se indica mediante su nombre en los segmentos *ARQ* y *AIP*.
El mensaje de respuesta utiliza la siguiente estructura:

```
MSH|^~\&|Their System|Their Remote Facility|LOCAL|LOCAL|<
    date>||SRR^S01^SRR_S01|<id>|P|2.5
MSA|<ack_code>|<answers_to>|<error_reason>?
SCH||||||timeslotQuery||||||||||integradorQuery||||
    integradorQuery
TQ1|1||||||<start_date>|<end_date>
RGS|1
```

En este mensaje se devuelve la información temporal acerca del horario reservado. Si la reserva se ha efectuado correctamente el *ack_code* tomará el valor *AA* y el campo de *error_reason* permanecerá vacío. En caso de que haya algún error, el campo *ack_code* tendrá el valor *AE* y el campo *error_reason* contendrá el motivo del error. Además, el segmento *TQ1* contiene información sobre el inicio y el fin de la cita reservada. El error que puede aparecer durante el procesamiento de este mensaje es que el hueco que se quiere no exista o bien ya haya sido reservado por otro usuario.

F.3. SRM_S02 y SRR_S02

Estos mensajes se utilizan para cubrir el caso de uso de modificación de una cita ya existente. En el caso de la petición de modificación de una cita, se utiliza el mensaje *SRM_S02*. Este mensaje tiene la siguiente estructura:

```
MSH|^~\&|LOCAL|LOCAL|Their System|Their Remote Facility|<
    date>||SRM^S02^SRM_S01|<id>|P|2.5
ARQ|<uid_cita>|<uid_cita>|||||<appointment_reason>||||<
    new_start_date>||||<profesional>||||^^integrador
PID|||^^^^<patient_uuid>||<patient_name>||||||<
    patient_address>
RGS|1
AIS|1|A|<service_id>^<service_name>
AIL|1|A|^^^&<service_location>
AIP|1|A|^<profesional>
```

Se puede observar que la estructura de este mensaje es igual a la del mensaje *SRM_S01* y por lo tanto en el segmento *MSH* se observa que aparece este tipo de mensaje. Además, en el segmento *ARQ* aparece la nueva fecha de inicio de la cita y los demás campos se mantienen igual a los usados para el mensaje de reserva de cita (Sección F.2).
La respuesta a este mensaje es un mensaje *SRR_S02* y, como en el caso anterior, comparte estructura con el mensaje *SRR_S01*. La estructura del mensaje de respuesta es la siguiente:

```
MSH|^~\&|Their System|Their Remote Facility|LOCAL|LOCAL|<
    date>||SRR^S02^SRR_S01|<id>|P|2.5
MSA|<ack_code>|<answers_to>|<error_reason>?
SCH||||||timeslotQuery||||||||||integradorQuery||||
    integradorQuery
TQ1|1||||||<start_date>|<end_date>
RGS|1
```

Como ocurría en el caso del mensaje de respuesta de la Sección F.2, aparecen las fechas iniciales y finales de la cita reservada. Además, se utiliza de igual manera el *ack_code* para determinar si la petición ha sido un éxito (AA) o ha ocurrido algún error (AE). Si ha ocurrido algún error, no se enviaría la información de la cita modificada y aparecería el mensaje contenido en *error_reason*. Los errores que pueden ocurrir en el procesamiento de este mensaje son los siguientes:

- El hueco que se quería reservar ya ha sido reservado.
- El paciente de la petición de la cita modificada no coincide con el que estaba originalmente en la cita.
- El servicio de la petición de la cita modificada no coincide con el que estaba originalmente en la cita.

F.4. SRM_S04 y SRR_S04

Los mensajes *SRM_S04* y *SRR_S04* se corresponden con los mensajes de petición de cancelación de cita y su respuesta. La estructura del mensaje de petición es la siguiente:

```
MSH|^~\&|LOCAL|LOCAL|Their System|Their Remote Facility|<
    date>||SRM^S04^SRM_S01|<id>|P|2.5
ARQ|<appointment_uuid>|<appointment_uuid>|||||||||<
    appointment_start_date>||||<profesional>||||^^
    integrador
```

En este mensaje se puede observar que se envía la información necesaria para buscar la cita que estaba reservada y poder realizar el borrado de la misma. Además, como en los casos anteriores el mensaje tiene la estructura del mensaje de reserva de cita *SRM_S01*.
La respuesta del mensaje de cancelación es la siguiente:

```
MSH|^~\&|Their System|Their Remote Facility|LOCAL|LOCAL|<
    date>||SRR^S04^SRR_S01|<id>|P|2.5
MSA|<ack_code>|<answer_to>|<error_reason>
```

Como ocurría en los mensajes anteriores, el *ack_code* puede tomar valores *AA* si se ha completado la petición correctamente o *AE* en caso de que haya ocurrido algún error. Además, si ha ocurrido algún error, el campo *error_reason* contendrá el motivo del error. El error que puede ocurrir durante el procesamiento de este mensaje, es que la cita con el identificador *appointment_uuid* no exista en el sistema HIS.

Bibliografía

[1] OpenMRS Wiki. Hl7. https://wiki.openmrs.org/display/docs/HL7. Consultado el 31/4/2021.

[2] Alistair Cockbur. Hexagonal architecture. https://alistair.cockburn.us/hexagonal-architecture/, Enero 2005. Consultado el 9/4/2021.

[3] Refactoring Guru. Abstract factory. https://refactoring.guru/es/design-patterns/abstract-factory. Consultado el 6/4/2021.

[4] Creating a custom rest service in mirth 3.0.x - mirth connect - confluence. http://www.mirthcorp.com/community/wiki/display/mirth/Creating+a+custom+REST+Service+in+Mirth+3.0.x. Consultado el 1/4/2021.

[5] NextGen Healthcare. Nextgen connect user guide for version 3.9. https://www.nextgen.com/-/media/files/nextgen-connect/nextgen-connect-39-user-guide.pdf, Abril 2020. Consultado el 3/4/2021.

[6] HL7 International. Resource patient - content. https://www.hl7.org/fhir/patient.html. Consultado el 4/4/2021.

[7] OpenMRS Wiki. Openmrs fhir module. https://wiki.openmrs.org/display/projects/OpenMRS+FHIR+Module#OpenMRSFHIRModule-FHIRDevelopment, Mayo 2021. Consultado el 6/4/2021.

[8] OpenMRS Wiki. Rest web services api for clients - authentication. https://wiki.openmrs.org/display/docs/REST+Web+Services+API+For+Clients#RESTWebServicesAPIForClients-Authentication, Abril 2020. Consultado el 5/4/2021.

[9] Microsoft. Tutorial: Uso de la inserción de dependencias en .net. https://docs.microsoft.com/es-es/dotnet/core/extensions/dependency-injection-usage, Abril 2021. Consultado el 10/5/2021.

[10] Microsoft. Llamada a una api web desde un cliente .net (c#). https://docs.microsoft.com/es-es/aspnet/web-api/overview/advanced/calling-a-web-api-from-a-net-client, Noviembre 2017. Consultado el 10/5/2021.

[11] Microsoft. Configuración en asp.net core. https://docs.microsoft.com/es-es/aspnet/core/fundamentals/configuration/?view=aspnetcore-5.0, Enero 2021. Consultado el 18/5/2021.

[12] Microsoft. Explore estos tutoriales para obtener información sobre las herramientas de .net y el sdk de .net. https://docs.microsoft.com/es-es/dotnet/core/tutorials/, Diciembre 2020. Consultado el 20/5/2021.

[13] OpenMRS. Openmrs rest api. https://rest.openmrs.org/#current-version. Consultado el 15/3/2021.

[14] Mirth Community. What is the use of filter and transformer. https://forums.mirthproject.io/forum/mirth-connect/support/8887-what-is-the-use-of-filter-and-transformer?p=62034#post62034, Octubre 2010. Consultado el 3/4/2021.

[15] Gerencia Regional de Salud de la Junta de Castilla y León. Guía de mensajería para gestión de citas. https://www.fundacionsigno.com/bazar/5/2_Mensajeria_Citas_y_lista_espera.pdf, Febrero 2010. Consultado el 31/3/2021.